НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 Інженерія програмного забезпечення

на тему Програмно – апаратний комплекс системи контролю управління доступом

Виконав: студент 4 курсу, групи ТІ-61

Висовень Денис Дмитрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н. Ковальчук А. М. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент доцент кафедри ТЕУ Т Та АЕС Сірий О.А _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення веб-технологій та мобільних пристроїв

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Висовень Денис Дмитрович_____
(прізвище, ім’я, по батькові)

1. Тема роботи ___Програмно – апаратний комплекс системи контролю управління доступом_____
_____

керівник роботи_____доцент, к.т.н. Ковальчук А. М._____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 202__р. № __

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: персональний комп’ютер, мови програмування C++, Java.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)__проаналізувати існуючі програмні та апаратні рішення, можливі методи реалізації взаємодії елементів, обґрунтувати обрані програмні застосунки та шляхи розробки програмного забезпечення, розробити необхідне програмне та апаратне забезпечення, розробити інтерфейс користувача, зробити висновки за результатами роботи._____

5. Перелік ілюстративного матеріалу

__1. Моделювання системи 2. Апаратна архітектура системи. 3.Програмна архітектура системи. 4. Поєднання апаратних засобів системи. 5. Демонстрація роботи програмних та апаратних засобів системи._____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”_2_” __грудня_____2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	2.12.2019	
2.	Вивчення та аналіз задачі	2.12.2019-13.04.2020	
3.	Розробка архітектури та загальної структури системи	13.04.2020-27.04.2020	
4.	Розробка структур окремих підсистем	27.04.2020-04.05.2020	
5.	Програмна реалізація системи	20.04.2020-13.05.2020	
6.	Оформлення пояснювальної записки	04.05.2020-05.06.2020	
7.	Захист програмного продукту	17.05.2020	
8.	Передзахист	12.06.2020	
9.	Захист	15.06.2020	

Студент _____ _____Висовень Д.Д._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Ковальчук А.М._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Мета роботи - розробка апаратно-програмного комплексу системи контролю та управління доступом. У створенні апаратного комплексу були використані апаратні платформи OrangePi PC, Arduino Nano. Для розробки програмного забезпечення було використано мову C++ (для контролерів Atmega), мову Java та фреймворки Spring Boot та Vaadin Framework.

Розроблений програмно-апаратний комплекс забезпечує контроль та управління доступом авторизованих та неавторизованих осіб до приміщень та територій, контрольованих системою.

Записка містить 84 сторінки, 22 рисунки, 6 таблиць, 3 додатки і 25 посилань.

Ключові слова: мульти-агентні системи, інтелектуальний агент, система контролю та управління доступом, охорона та безпека, розподілені системи.

АННОТАЦИЯ

Цель работы - разработка программно-аппаратного комплекса системы контроля и управления доступом. В создании аппаратного комплекса были использованы аппаратные платформы OrangePi PC, Arduino Nano. Для разработки программного обеспечения были использованы язык C ++ (для контроллеров Atmega), язык Java и фреймворки Spring Boot и Vaadin Framework.

Разработанный программно-аппаратный комплекс обеспечивает контроль и управление доступом авторизованных и неавторизованных лиц в помещения и территорий, контролируемых системой.

Записка содержит 84 страницы, 22 рисунков, 6 таблиц, 3 приложения и 25 ссылок.

Ключевые слова: мульти-агентные системы, интеллектуальный агент, система контроля и управления доступом, охрана и безопасность, распределённые системы.

ABSTRACT

The purpose of the work is to develop a hardware-software complex of access control system. OrangePi and Arduino Nano were used in the creation of the hardware complex. C++ (for Atmega controller), Java with Spring Boot and Vaadin Framework were used to develop the software.

The developed hardware-software complex provides control of access of authrorised and unauthorized personnel to rooms and territories, controlled by the system.

Note contains 84 pages, 22 drawings, 6 tables, 3 attachments and 25 links.

Key words: multi-agent systems, intellectual agent, access control system, security, distributed systems.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

TCP/IP – Transmission Control Protocol/Internet Protocol

UART – Universal Asynchronous Receiver/Transmitter

GPIO – General Purpose Input Output

PWM – Pulse Width Modulation

RAM – Random Access Memory

SRAM – Static RAM

EEPROM – Electrically Erasable Programmable Read-Only Memory

USB – Universal Serial Bus

ORM – Object Relational Mapping

API – Application Programming Interface

DTO – Data Transfer Object

БД – База Даних

СУБД – Система Управління Базою Даних

SQL – Script Query Language

JDBC – Java Database Connector

ACS – Access Control System

СКУД – Система контролю та управління доступом

EE – Enterprise Edition

NFC – Near Field Communication

MQTT – Message Queuing Telemetry Transport

IDE – Integrated Development Environment

IoT – Internet of Things

ОС – Операційна Система

SSH – Secure Shell

VGA – Video Graphics Array

HDMI – High Definition Multimedia Interface

API – Application Programming Interface

CAD – Computer Aided Design

XML – eXtensible Markup Language

MVC – Model View Controller

DAO – Data Access Object

ООП – Об’єктно орієнтоване програмування

SE – Standart Edition

DVFS - Dynamic Voltage and Frequency Scaling

JMS – Java Messaging System

JCA – Java EE Connector Architecture

JMX – Java Management Extensions

UML – Unified Modeling Language

ООП – Об’єктно Орієнтоване Програмування

ВСТУП

Розробка відмовостійкого програмно-апаратного комплексу, який буде відповідати високим вимогам, які застосовуються до систем контролю та управління доступом, потребує значних зусиль при проектуванні та моделюванні майбутньої системи. Крім загальних вимог з безпеки та надійності, на таку систему накладаються додаткові вимоги з швидкості роботи системи та простоти у адмініструванні. Крім цього, встановлення та обслуговування системи не повинно вимагати заміни значної частини елементів системи та виконання цього великою кількістю висококваліфікованих спеціалістів.

Найбільшу складність у розробці подібної системи створює продумування систем комунікації елементів, з можливістю подальшого розвитку системи. Системи з незначними можливостями до адаптації можуть ставати непорушними монолітами, у разі зміни навколишньої ситуації. Протокол та методи спілкування у подібних системах повинні забезпечувати гнучкість, та простоту адаптації різних програмних та апаратних засобів для виконання встановлених перед системою цілей.

Найбільш ефективним рішенням проблеми монолітності системи є концепція мульти-агентних системи. Мульти-агентні системи - це сукупність незалежних інтелектуальних агентів, кожен з яких контролює лише підвладну йому ланку[1]. Але, для відповідності стандартам СКУД, концепція має використовуватись з незначними змінами. Для мульти-агентних систем є характерними автономність кожного агенту, обмеженість уявлення агенту про його систему та відсутність головних керуючих агентів[1]. Головною зміною у цю концепцію буде внесення умовно головного інтелектуального агента системи – йому буде відомо про стан кожного елемента системи. Таких змін вимагає специфіка роботи охоронних систем. Завдяки такому підходу до проектування системи, забезпечує надійну та безпечну систему, у якій помилка під час роботи одного інтелектуального агенту мінімально впливає на роботу усієї системи.

Виходячи з цього, було запропоновано дослідити актуальний ринок систем контролю та управління доступом, а також їх користувачів, дослідити різні апаратні

платформи та технології розробки програмного забезпечення розподілених систем, зокрема мульти-агентних систем, та, у результаті, створити гнучкий програмно-апаратний комплекс системи контролю та управління доступом.

Було поставлено завдання: проаналізувати різні СКУД, спланувати систему, використовуючи концепцію мульти-агентних систем, розробити програмну та апаратну частини агентів системи, розробити інтерфейс користувача, який дозволить користуватись системою.

Для реалізації даного програмного продукту було обрано фреймворки Spring Boot, Vaadin Framework та мову програмування Java.

1. АНАЛІЗ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ МУЛЬТИ-АГЕНТНОЇ СИСТЕМИ КОНТРОЛЮ ТА УПРАВЛІННЯ ДОСТУПОМ

В умовах жорсткої конкурентної боротьби на ринку нових технологій, дотримання умов секретності та протистояння витоку технологічної інформації, інформаційної безпеки, несанкціонованого доступу до обладнання та території для деяких підприємств критичним постає питання організації та контролю доступу в технологічні зони та приміщення [4].

У даній роботі розглядається проблема адаптації системи контролю та управління доступом як інтелектуального агенту мульти-агентної системи [5]. Це також дозволяє створювати сценарії роботи системи контролю та управління в умовах мінливості ситуації.

Адаптація системи контролю та управління доступом полягає у значному покращенні комунікації та функціоналу її складових для підвищення гнучкості системи при роботі з різними типами вимог та зовнішніх факторів [6].

Сучасні системи контролю та управління доступом є типовими закритими системами без можливості швидкої адаптації до локальних вимог, а також не дозволяє зовнішнім чинникам впливати на внутрішні регламенти роботи системи.

Дана робота описує створення агенту розподіленої системи контролю та управління доступом до технологічних зон та приміщень (далі СКУД). Її роль полягає у забезпеченні регламенту авторизованого та неавторизованого доступу та переміщення осіб в зоні контролю.

Для забезпечення високого рівня захищеності об’єкта від несанкціонованого втручання в умовах мінливості ситуації необхідно враховувати [7, 8]:

1. Можливість зміни регламентів доступу протягом доби.
2. Можливість зміни регламентів доступу в разі виникнення екстрених ситуацій (пожежа, аварія, теракт, стихійне лихо, тощо).
3. Можливість зміни регламентів доступу до обладнання, його активації та/або зміни режиму функціонування, відповідно до рівня допуску.

4. Можливість моніторингу переміщень осіб (та їх правомірності) в зоні контролю.

1.1 Опис системи контролю та управління доступом

Сучасний ринок СКУД має низький рівень розвитку та інновацій.

В існуючій літературі про сучасні системи управління та контролю доступу описані наступні принципи роботи СКУД [4, 7]:

- протидія промисловому шпигунству;
- протиугінна дія;
- дія проти диверсій;
- від навмисного збитку матеріальних цінностей;
- відстеження часу;
- вчасно контролювати прибуття та від'їзд працівників;
- захист конфіденційності інформації;
- регулювання потоку відвідувачів;
- контроль в'їзду та виїзду транспорту та вантажу.

Крім того, СКУД є бар'єром для "допитливих" людей [4].

При впровадженні конкретних систем контролю доступу використовуються різні методи та пристрої для ідентифікації та перевірки особи.

Як найбільш часто використовуваних СКУД можна назвати такі [4]:

- турнікети звичайні і настінні;
- турнікети для проходу в коридорах;
- шлюзові кабіни;
- автоматичні хвіртки;
- роторні турнікети;
- обертові двері;
- дорожні блокіратори;
- шлагбауми;
- паркувальні системи;

- круглі розсувні двері;
- трехштанговие турнікети;
- повнозростові турнікети;
- розсувні турнікети.

Дуже важливим питанням є можливість інтеграції СКУД до будь-якої системи безпеки за допомогою відкритого протоколу [4].

Основними завданнями контрольно-пропускних пунктів є [4, 7, 8]:

- захист законних інтересів підприємства, підтримка внутрішнього управління;
- захист власності бізнесу, його раціональне та ефективне використання;
- збільшення прибутку підприємств;
- внутрішня та зовнішня стабільність організації;
- захист комерційної таємниці та прав інтелектуальної власності.

Контрольно-пропускний пункт як частина системи безпеки дозволяє власнику вирішувати такі завдання [4, 7, 8]:

- Забезпечення авторизованого проходу працівників та відвідувачів, ввезення / вивезення матеріальних продуктів та активів, роздрібнення підприємства;
- Запобігання неконтрольованому проникненню сторонніх осіб та транспортних засобів у заборонені зони та приватні будівлі (приміщення);
- Своєчасне виявлення загроз інтересам підприємства, а також потенційно небезпечних умов, які можуть заподіяти матеріальну та моральну шкоду підприємству;
- Створення надійних гарантій для підтримки організаційної стабільності зовнішніх та внутрішніх зв’язків підприємства, розробка механізму швидкого реагування на загрози та негативні тенденції;
- Придушення проблем у законних інтересах підприємства, використання правових, економічних, організаційних, соціально-психологічних, технічних та інших засобів для виявлення та зменшення джерел загрози безпеці підприємства.

Контрольно-пропускні пункти можна визначити як систему надання регуляторних, організаційних та матеріальних гарантій для виявлення, запобігання та протидії порушенню законних прав підприємства, його власності, інтелектуальної власності, виробничої дисципліни, технологічного керівництва, наукових досягнень та захищеної інформації; як сукупність організаційно-правових обмежень та правил, що встановлюють порядок проходження через пункт пропуску працівників закладу, відвідувачів, транспортування імпорту / експорту матеріальних цінностей [4].

1.2 Опис мульти-агентної системи

Агент, у рамках архітектури мульти-агентної системи – це комп'ютерна система, що розташована в певному середовищі, і може автономно працювати в цьому середовищі для досягнення закладених в неї цілей.

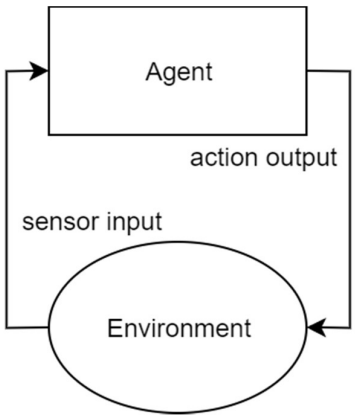

Рисунок 1.1 Зображення типової схеми роботи агенту мульти-агентної системи

На рисунку 1.1 представлено абстрактний вигляд інтелектуального агенту. На цьому рисунку ми можемо побачити вихід дії, згенерований агентом з метою впливу на його оточення. У більшості областей розумної складності агент не матиме повного контролю над своїм оточенням. Він матиме, в кращому випадку, частковий контроль, оскільки може впливати на це. З точки зору агенту, це означає, що одна і та ж дія, що виконується двічі за ідентичних зовнішніх обставин, може мати абсолютно різні наслідки, і, зокрема, вона може не мати бажаного ефекту. Тому агенти слід використовувати в інших середовищах, крім найбільш вірогідних для можливості виходу з ладу. Ми формально можемо підсумувати ситуацію, сказавши, що

навколишнє середовище взагалі вважається недетермінованим. Зазвичай агент здійснює різні дії, доступні йому. Цей набір можливих дій представляє його здатність змінювати своє оточення. Зауважимо, що не всі дії можна виконувати у будь-яких ситуаціях. Наприклад, дію "підйому столу " можна застосовати лише в тих випадках, коли вага столу достатньо мала, щоб агент міг його підняти. Аналогічно, дія «придбати Ferrari» буде невдалою, якщо для цього не вистачить коштів. Тому операції мають передумови, які визначають можливі ситуації, в яких вони можуть бути застосовані. Основна проблема, з якою стикається агент, - це вирішити, які його дії він повинен виконувати, щоб найкращим чином задовольнити свої цілі дизайну. Агентські архітектури – це програмні архітектури для систем прийняття рішень, які вбудовані в середовище [1, 2, 5, 6].

Систему управління можна розглядати як агент. Простий приклад такої системи – термостат. Термостати мають датчик для визначення температури в приміщенні. Цей датчик вбудований безпосередньо в навколишнє середовище (тобто приміщення), і він видає як вихід один з двох сигналів: один, який вказує, що температура занадто низька, інший, який вказує, що температура перебуває у визначених межах. Дії, які доступні терморегулятору – це «ввімкнути нагрівання» або «вимкнути нагрівання». Дія «ввімкнути нагрівання», як правило, впливає на підвищення температури в приміщенні, але це не може бути гарантованим ефектом – якщо, наприклад, двері в приміщення відчинені, включення обігрівача може не мати ніякого ефекту. Спрощений компонент термостату для прийняття рішень реалізує (як правило, електромеханічне обладнання) такі правила [1, 3, 6]:

- занадто холодно - ввімкнути нагрівання;
- температура в порядку - вимкнути нагрівання.

Більш складні системи контролю навколишнього середовища, звичайно, мають значно масштабнішу систему прийняття рішень. Прикладом таких рішень можуть бути автономні космічні зонди, літальні апарати, системи управління ядерними реакторами тощо [1, 5].

Основні умови інтелектуальності агентів [1]:

- Реактивність. Інтелектуальні агенти здатні сприймати своє оточення та своєчасно реагувати на зміни, які в ньому відбуваються, щоб виконати поставлені задачі.
- Проактивність. Розумні агенти здатні проявляти цілеспрямовану поведінку, проявляючи ініціативу, щоб виконати поставлені задачі.
- Соціальність. Розумні агенти здатні взаємодіяти з іншими агентами (і, можливо, людьми), щоб виконати поставлені задачі.

2 АРХІТЕКТУРА АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ

Використання концепції мульти-агентних систем у СКУД є новацією, що дозволяє вирішувати задачі, які раніше вважались неможливими у процесах регулювання переміщення авторизованих та неавторизованих осіб у контрольованих системою зонах.

Але, для використання такого підходу, потрібно визначити архітектуру системи, з виконанням вимог до СКУД, для запобігання несанкціонованого доступу та порушення регламенту доступу, а також для запобігання людських жертв під час надзвичайних ситуацій.

2.1 Архітектура мульти-агентної системи

На рисунку 2.1 зображено сукупність елементів СКУД, згрупованих у мульти-агентну систему. Для забезпечення надійності системи її функціональні задачі були розділено на незалежні частини-агенти.

Робота присвячена розробці мульти-агентної СКУД. До неї входять декілька агентів, що виконують весь спектр задач СКУД. Найважливішим є адміністративний агент, головним завданням якого є моніторинг стану інших агентів та забезпечення їх актуальним регламентом роботи системи.

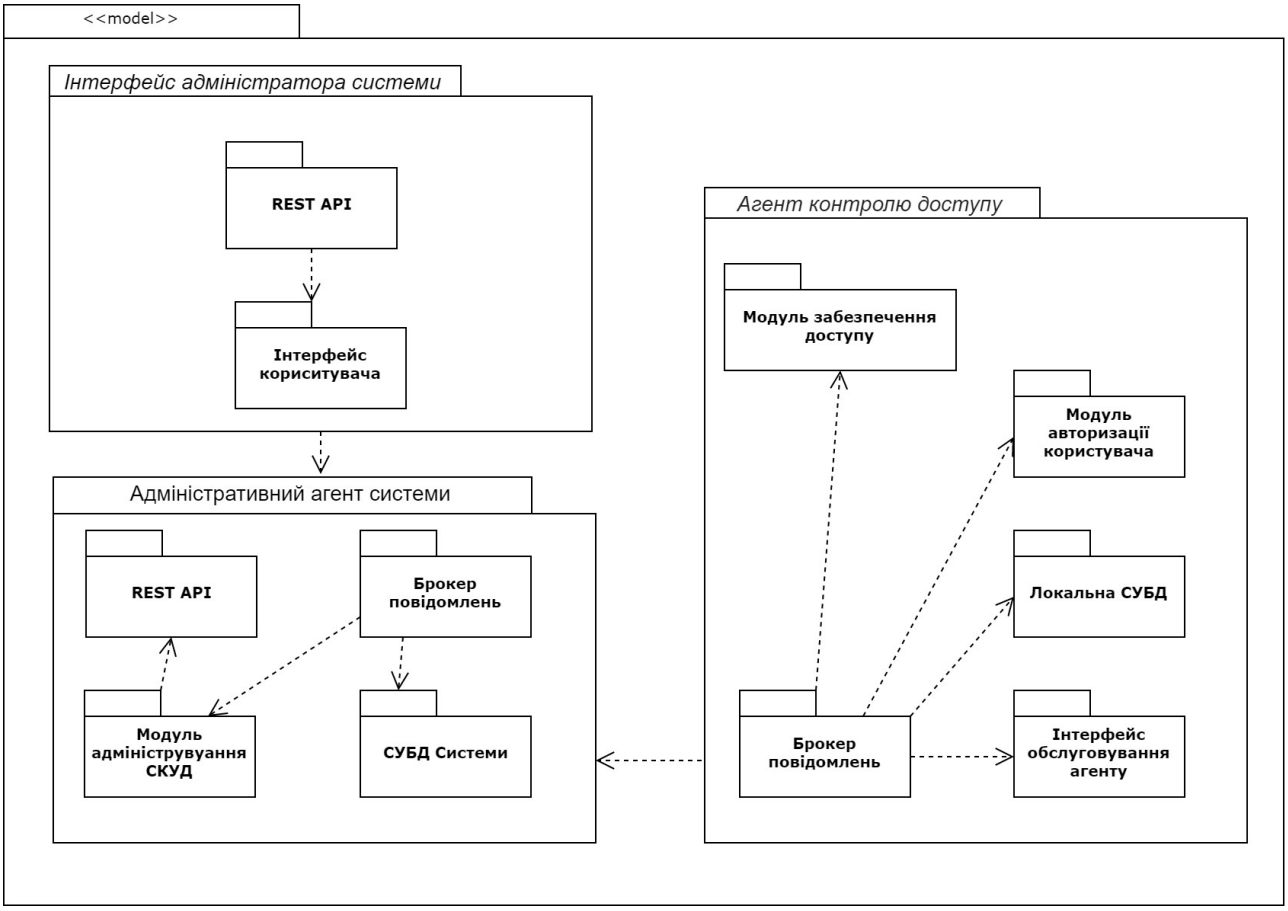

Рисунок 2.1 – Мульти-агентна СКУД.

У системі також присутні інші інтелектуальні агенти, кожен з яких виконує свою функцію і може бути заміненим або доповненим іншим інтелектуальним агентом. Наприклад, агент контролю доступу забезпечує всіх авторизованих користувачів системи доступом до контрольованого ним приміщення. У той же час, він інформує всіх інших агентів що авторизований користувач переміщується та стежить за змінами у регламенті роботи системи. Агент адміністрування системи стежить за переміщенням авторизованих користувачів та може корегувати регламент роботи системи відповідно до навколишніх умов або за втручання адміністратора системи. При цьому, адміністратор може вносити зміни у регламент доступу використовуючи агент інтерфейсу адміністратора.

Комунікація між агентами мульти-агентної системи забезпечується протоколом обміну інформацією на базі на базі протоколу брокеру повідомлень MQTT.

Для зв’язку між агентами мережі, кожен агент має мати з’єднання з відповідним сервером брокеру повідомлень для роботи елемента комунікації агенту.

2.1.1 Архітектура агенту контролю доступу

Агент контролю доступу складається з декількох основних функціональних елементів:

- Елемент забезпечення доступу
- Елемент авторизації користувача системи
- Локальна СУБД
- Веб-інтерфейс для обслуговування агенту
- Елемент комунікації з іншими агентами

Елемент забезпечення доступу регулює роботу апаратного модуля забезпечення доступу, що може бути електромагнітним замком, турнікетом, тощо. Якщо регламент дозволяє користувачу системи у даний момент отримати доступ до відповідного контрольованого приміщення, або території, то цей елемент впливає на свою апаратну складову для надання відповідного доступу користувачу.

Елемент авторизації користувача системи стежить за роботою апаратного модуля авторизації користувача, що може бути кодовою панеллю, або NFC-зчитувачем, тощо. У разі використання користувачем апаратного модуля, цей елемент перевіряє свою локальну СУБД на подібні дані автентифікації і якщо знаходить там користувача, то повідомляє елемент забезпечення доступу про користувача, що хоче отримати доступ.

Локальна СУБД зберігає у собі поточний регламент роботи цього агенту СКУД та базу даних користувачів цього агенту мульти-агентної системи. Ця СУБД оновлюється при кожній зміні регламенту роботи агенту, або зміні даних користувачів цього агенту, отримуючи актуальну інформацію від адміністративного агенту системи.

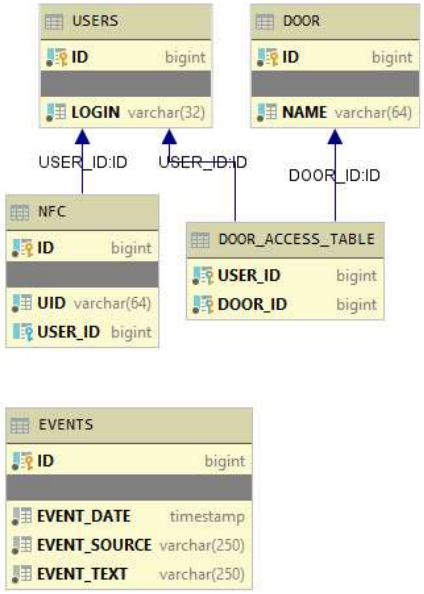

Рисунок 2.2 – Структура локальної СУБД агенту контролю доступу

Структуру глобальної СУБД зображено на рисунку 2.2.

Веб-інтерфейс обслуговування агенту дозволяє перевіряти поточний стан агенту та його історію роботи. Ця інформація може бути використана для дослідження причин неправильної роботи регламенту СКУД, або цього конкретного інтелектуального агенту.

Комунікація з іншими агентами мульти-агентної системи забезпечується протоколом обміну інформацією на базі на базі протоколу брокеру повідомлень MQTT.

2.1.2 Архітектура адміністративного агенту СКУД

Адміністративний агент СКУД складається з наступних функціональних частин:

- Елемент менеджменту СКУД
- Глобальна СУБД
- Елемент комунікації з іншими агентами

Елемент менеджменту СКУД займається моніторингом стану системи та її окремих агентів. Крім цього, він підтримує актуальними локальні СУБД на кожному

з агентів контролю доступу. Також, цей елемент відповідає за прийняття рішень відповідно до зміни навколишньої та внутрішньої поточної ситуації.

Глобальна СУБД зберігає повну версію регламенту роботи СКУД та повну базу користувачів системи.

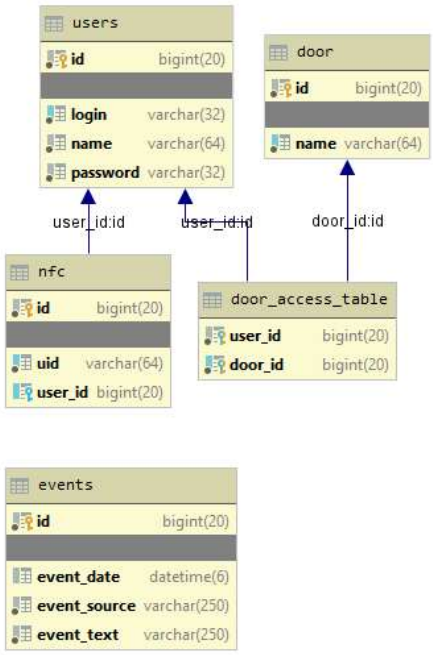

Рисунок 2.3 – Структура глобальної СУБД СКУД

Структуру глобальної СУБД зображено на рисунку 2.3.

2.1.3 Архітектура агенту інтерфейсу адміністратора СКУД

Агент інтерфейсу адміністратора СКУД забезпечує адміністратора СКУД можливістю власноруч впливати на поточний регламент системи, редагувати користувачів та їх права доступу.

Основним його елементом виступає інтерфейс користувача, який відправляє сформовані команди до адміністративного агенту системи.

2.2 Архітектура апаратної частини агенту контролю доступу

Згідно з вимог до СКУД було створено апаратно-програмний комплекс, що виконує функції апаратної платформи для агенту контролю доступу мульти-агентної системи контролю та управління доступом.

Апаратна платформа складається з наступних основних елементів, що зображені на рисунку 2.4:

- Мікрокомп’ютер
- Модуль управління живленням
- Стабілізатор живлення
- Транзисторні ключі

Такий набір компонентів є основою надзвичайно гнучкої апаратної платформи.

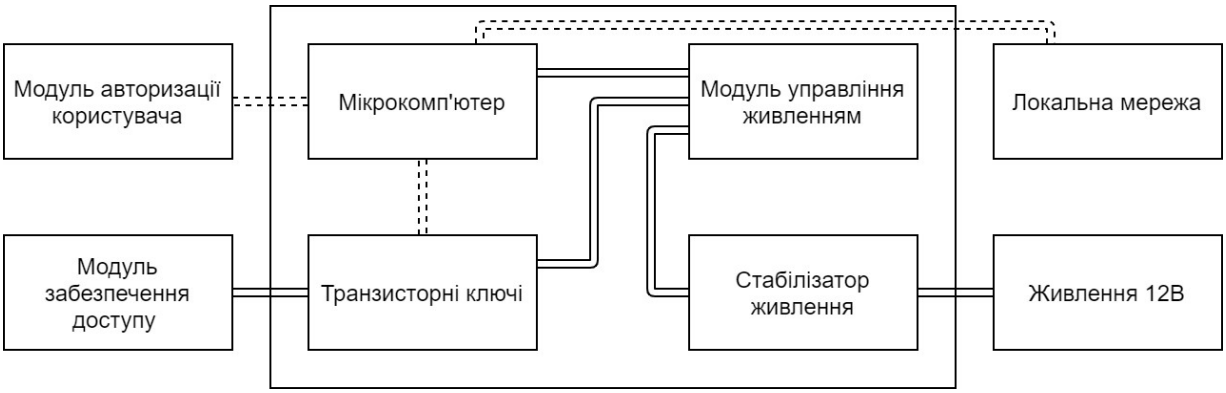

Рисунок 2.4 – Структура апаратної платформи агенту контролю доступу. Пунктиром позначено шини даних.

Головним завданням цієї апаратної платформи є забезпечення можливості гнучкої апаратної конфігурації інтелектуального агенту під конкретну задачу використовуючи найбільш відповідні компоненти.

Схема апаратної платформи зображена на рисунках 2.5 та 2.6.

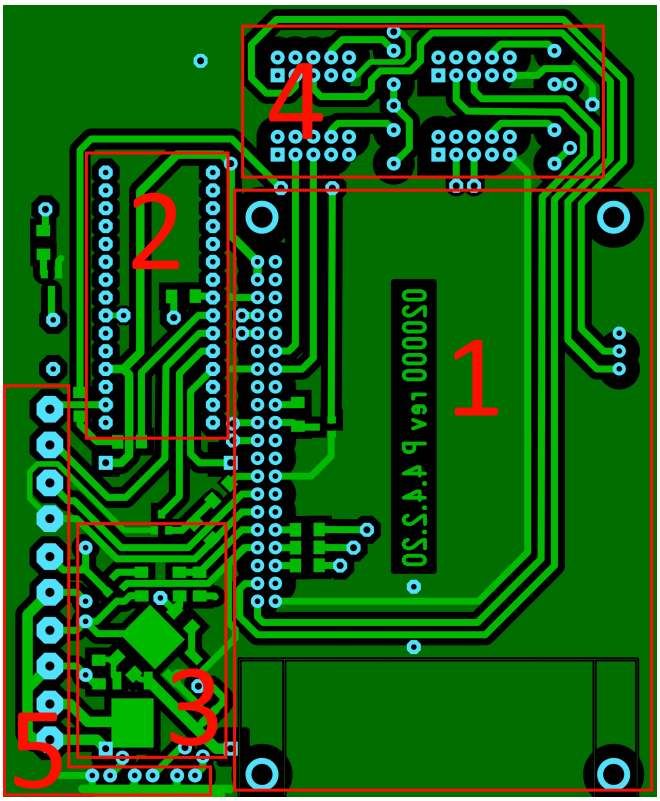

Рисунок 2.5 – Електрична схема апаратної платформи агенту контролю доступу, вид знизу

На рисунку 2.5 присутні наступні позначення, пронумеровані та виділені червоним:

1. Мікрокомп’ютер
2. Модуль управління живленням
3. Транзисторні ключі
4. Роз’єми для підключення DB9 UART виводів
5. Роз’єми для підключення живлення 12В, світлодіодів, кнопки ввімкнення та вимкнення та модулю забезпечення доступу

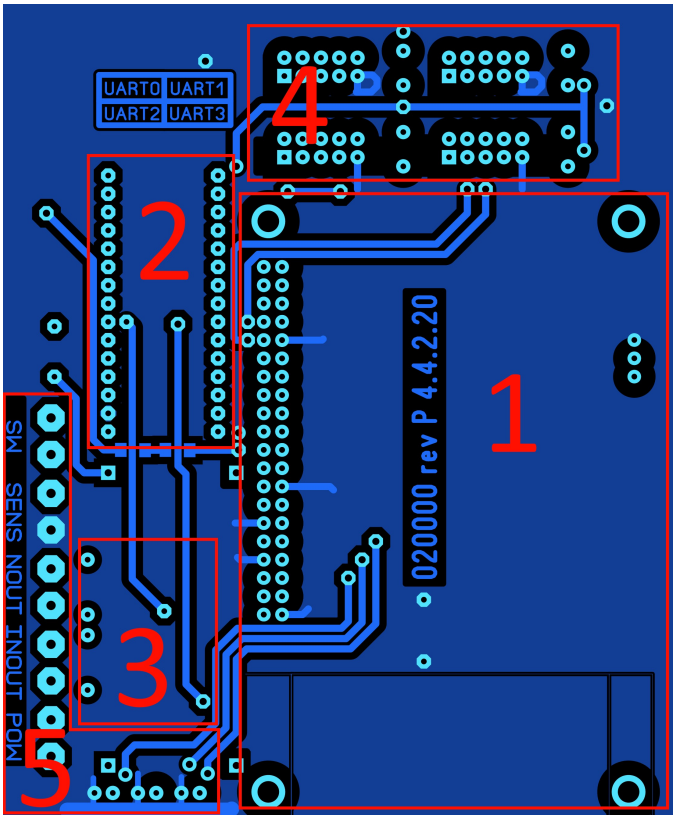

Рисунок 2.6 – Електрична схема апаратної платформи агенту контролю доступу, вид зверху

На рисунку 2.6 присутні наступні позначення, пронумеровані та виділені червоним:

1. Мікрокомп’ютер
2. Модуль управління живленням
3. Стабілізатор живлення
4. Коннектори для підключення DB9 UART виводів
5. Роз’єми для підключення живлення 12В, світлодіодів, кнопки ввімкнення та вимкнення та модулю забезпечення доступу

Мікрокомп’ютер є серцем апаратної платформи. Він контролює роботу пристрою, та саме на ньому працює програмне забезпечення агенту контролю доступу СКУД.

Транзисторні ключі з різними виводами дозволяють підключити до апаратної платформи широкий спектр різних апаратних засобів забезпечення доступу. Це може бути як звичайний електронно-механічний замок, так і турнікет, або електромагніт.

Модуль авторизації користувача підключається до апаратної платформи використовуючи роз'єм DB9 та спілкується з мікрокомп'ютером за протоколом UART і подібних з'єднань допустимо водночас 4 штуки. Використання такого типу з'єднання дозволяє використовувати різні способи авторизації користувача. Наприклад, можна підключити водночас матричну клавіатуру, біометричний сканер та NFC-зчитувач, і таким чином використовувати декілька різних методів для автентифікації користувача системи.

Для спрощення задачі живлення апаратної платформи, її слід підключати до живлення постійного току 12В. Це дозволяє не тільки використовувати 12В блоки живлення, але і застосовувати блоки безперебійного живлення. Використання 12В живлення дозволяє також використовувати таке живлення і для різних модулів забезпечення доступу, у яких 12В вважається стандартом. Мікрокомп'ютер у цій апаратній платформі дуже чутливий до свого джерела живлення, тому між входом 12В та мікрокомп'ютером встановлено ще два компоненти – модуль управління живленням та стабілізатор живлення.

Стабілізатор живлення дозволяє змінити напругу на вході до необхідних показників. Так, у цій платформі, стабілізатор живлення відрегульовано постачати 5В постійного струму, замість 12В.

Модуль управління живленням працює у парі з стабілізатором. Його завданням є регулювання постачання живлення до мікрокомп'ютера, його коректне ввімкнення та вимкнення.

Принцип взаємодії елементів апаратної платформи зображено на рисунку 2.7.

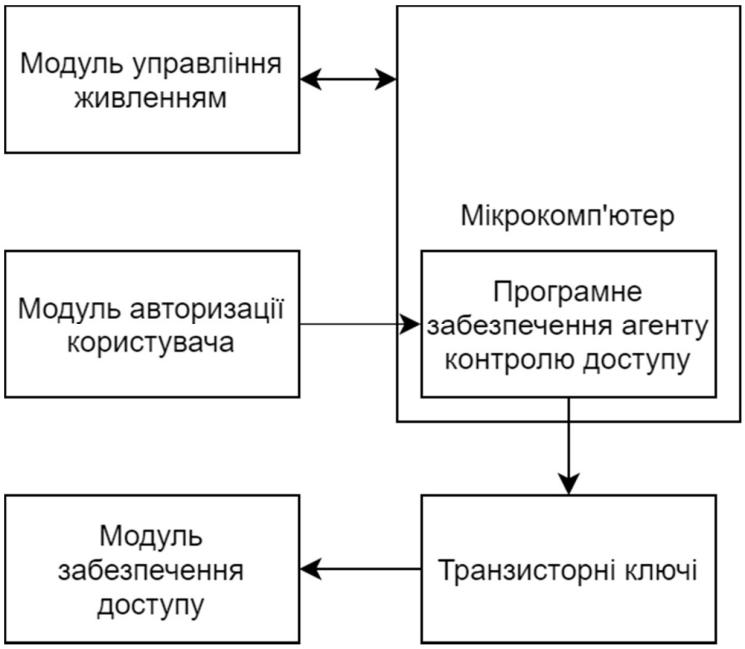

Рисунок 2.7 – Структура взаємодії елементів апаратної платформи

Така архітектура дозволяє вільно змінювати складові апаратної платформи, дозволяючи легко дробляти або реконфігурувати її для виконання поставлених на неї задач.

2.3 Архітектура програмної частини агенту контролю доступу

Програмна частина агенту контролю доступу складається з наступних частин:

- Контролер СУБД
- Контролер комунікації за протоколом UART
- Контролер інтерфейсу обслуговування
- Контролер управління елементом забезпечення доступу
- Контролер GPIO взаємодії
- Контролер зовнішнього зв’язку агенту

Основна задача агенту контролю доступу – забезпечувати доступ до контрольованих приміщень та територій для авторизованого та неавторизованого персоналу за регламентом системи. Виходячи з цього, важливо розділити алгоритм роботи агенту на максимальну кількість незалежних частин, для спрощення їх розробки та модифікації.

Головна задача контролеру СУБД – забезпечувати швидкий доступ до даних авторизації користувачів та поточного регламенту роботи агенту. Причому, специфіка роботи програмно-апаратного комплексу вимагає вбудовувати БД у програмне забезпечення, що накладає додаткові вимоги при виборі СУБД. Для спрощення розробки та підвищення надійності взаємодії з СУБД використовується модель ORM.

Від контролеру комунікації за протоколом UART вимагається розпізнавати під'єднаний до нього вид зчитувача, та опрацьовувати інформацію що надходить відразу з декількох зчитувачів що можуть бути під'єднаними до апаратної платформи. У момент авторизації користувача, цей контролер викликає витягує з контролеру СУБД інформацію про цього користувача та відправляє її до контролеру управління елементом забезпечення доступу.

Контролер управління елементом забезпечення доступу постійно виконує досить просту задачу: отримати дані користувача, звірити з регламентом (який він отримує від контролеру СУБД) та подати команду контролеру GPIO у разі необхідності надати доступ.

Контролер GPIO взаємодії відповідає за обслуговування низкорівневої взаємодії зазначеної апаратної платформи. Через нього проходять виклики до контактів транзисторних ключів, модулю управління живленням та корпусних світлодіодів.

Контролер інтерфейсу обслуговування відповідає за функціонування веб-панелі обслуговування. Ця панель дозволяє обслуговуючому персоналу реконфігурувати роботу модуля через веб-інтерфейс, та переглядати історію роботи разом з списком помилок.

Контролер зовнішнього зв'язку агенту дозволяє цьому агенту спілкуватись з іншими агентами мережі. Саме через цей контролер агент отримує актуальну інформацію від агенту адміністрування СКУД. Таблицею 2.1 зазначено протокол комунікації, яким користується цей контролер.

Таблиця 2.1 – Протокол взаємодії агенту контролю доступу з іншими агентами

Команда	Структура команди	Значення
Door Auth	<name>*doorName*<name>	Авторизація агенту у мульти-агентній системі з

		визначеним ідентифікатором *doorName*
DB Request Update	<db><name>*doorName*	Запит на оновлення БД з визначеним ідентифікатором *doorName* до агенту адміністрування СКУД
DB Update	DoorDTO	Пакет для оновлення локальної БД агенту
Door Event	EventDTO	Пакет інформування інших агентів подіями агенту контролю доступу

Ці команди передаються між агентами використовуючи спеціальний програмний засіб – брокер повідомлень. Використання брокеру повідомлень спрощує процес розробки і, при цьому, забезпечує високий рівень надійності та безпеки при передачі даних.

2.4 Архітектура програмної частини адміністративного агенту СКУД

Програмна частина агенту контролю доступу складається з наступних частин:

- Контролер СУБД
- Контролер стану агентів контролю доступу
- Контролер REST API
- Контролер користувачів системи
- Контролер взаємодії з іншими агентами

Головним завданням цього агенту мульти-агентної системи є відстеження та оновлення стану інших агентів, контроль переміщення користувачів, забезпечення роботи інтерфейсу адміністратора та реагування на зміну навколишньої та внутрішньої ситуації мульти-агентної системи.

Основне завдання контролеру СУБД забезпечити швидкий доступ до регламенту роботи СКУД та історії подій системи. Для спрощення розробки та підвищення надійності взаємодії з СУБД використовується модель ORM.

Контролер REST API забезпечить можливості інтеграції інших програмних засобів до цієї мульти-агентної системи. Наприклад, це дозволяє створити мобільний додаток, що буде мати функціонал кабінету адміністратора. Цей інтерфейс забезпечує можливість використовуючи HTTPS запити доступатись до інформації всередині мульти-агентної системи та взаємодіяти з нею.

Таблиця 2.2 – REST API Протокол зовнішньої взаємодії з мульти-агентною системою

Запит	Вид запиту	Структура запиту	Результат
/api/getdoor/excluded/{userid}	GET	Userid – ідентифікатор користувача	Повертає список усіх дверей до яких у користувача немає доступу
/api/getdoor/{id}	GET	Id – ідентифікатор агенту контролю доступу	Повертає сутність агенту контролю доступу з БД
/api/getdoors	GET		Повертає список усіх дверей у системі
/api/getkey/{id}	GET	Id – ідентифікатор ключа авторизації користувача	Повертає данні ключа авторизації
/api/getuser/{id}	GET	Id – ідентифікатор користувача системи	Повертає данні користувача системи
/api/getusers	GET		Повертає данні всіх користувачів системи
/api/getusers/filtered/{name}	GET	Name – фільтр пошуку за іменем користувача	Повертає данні всіх користувачів системи за заданим фільтром
/api/view/nfc/update	POST	Сутність ключа авторизації користувача	Оновлює дані про ключа

			авторизації користувача
/api/view/nfc/{Id}/remove	POST	Id – ідентифікатор ключа авторизації користувача	Видалити ключ авторизації користувача з системи
/api/view/update-event	GET	LongPolling	Повідомлення про зміни у БД
/api/view/user/create	POST	Сутність користувача системи	Створює користувача СКУД
/api/view/user/update	POST	Сутність користувача системи	Оновлює дані про користувача системи
/api/view/user/{userId}/add-door/{doorId}	POST	userId – ідентифікатор користувача системи; doorId – ідентифікатор агенту контролю доступу	Додає користувача до регламенту доступу до визначеного агенту контролю доступу
/api/view/user/{Id}/create/nfc	POST	Id – ідентифікатор користувача системи; Сутність ключа авторизації користувача	Додає новий ключ авторизації до користувача
/api/view/user/{Id}/remove	POST	Id – ідентифікатор користувача системи	Видаляє користувача з СКУД
/api/view/user/{userId}/remove-door/{doorId}	POST	userId – ідентифікатор користувача системи; doorId – ідентифікатор агенту контролю доступу	Видаляє користувача з регламенту доступу до визначеного агенту контролю доступу

Для спрощення реалізації програмно-апаратного комплексу, інтерфейс адміністратора використовує REST API для взаємодії з системою.

Контролер стану агентів контролю доступу стежить за роботою агентів контролю доступу, підключає їх до системи та збирає історію їх роботи. Однією з

його основних задач є підтримка актуальності регламенту у агентах контролю доступу, а також списку користувачів що мають доступ у поточний час. Зміна регламенту роботи конкретного сегменту мульти-агентної системи вимагає оновлення цього регламенту на кожному з агентів контролю доступу у цьому сегменті мульти-агентної системи.

Контролер користувачів системи відстежує переміщення осіб у контрольованих системою зонах, використовуючи інформацію що надходить від агентів контролю доступу. Ця інформація може використовуватись іншими агентами мульти-агентної системи для виконання поставлених задач.

Контролер взаємодії з іншими агентами дозволяє адміністративному агенту обмінюватись запитами з іншими агентами та відповідати їм. Протокол спілкування між агентами описаний у таблиці 2.1. Для передачі повідомлень між агентами використовується спеціалізований програмний засіб – брокер повідомлень.

2.5 Архітектура мережі комунікації між агентами

Для взаємодії між собою, агенти використовують спеціалізоване програмне забезпечення – брокер повідомлень. Це програмне забезпечення слугує проміжною ланкою у взаємодії між інтелектуальними агентами, яка забезпечує надійність та безпеку передачі інформації між ними.

Для коректної роботи системи, кожен агент має декілька потоків даних. Структура цих потоків даних описана у таблиці 2.3. Протокол їхньої взаємодії вже був зазначений у таблиці 2.1.

Таблиця 2.3 – Структура відносин агентів та потоків даних у мульти-агентній системі

Назва потоку	Видавці даних	Слухачі	Значення
AdminIn	Кожен з агентів контролю доступу	Один з адміністративних агентів СКУД на кожен власний потік даних	Потік вводу даних до кожного агенту контролю доступу
AdminOut	Один з адміністративних агентів СКУД на кожен власний потік даних	Кожен з агентів контролю доступу	Потік виводу даних з кожного агенту контролю доступу

DoorIn	Кожен з адміністративних агентів СКУД	Один агент контролю доступу на кожен власний потік даних	Потік вводу даних до кожного агенту контролю доступу
DoorOut	Один агент контролю доступу на кожен власний потік даних	Кожен з адміністративних агентів СКУД	Потік виводу даних з кожного агенту контролю доступу

Взаємодія між агентами розпочинається з встановлення базового зв’язку між агентами. Для цього, агент контролю доступу у режимі Slave, по відношенню до Master у адміністративного агенту, відправляє на потік входу даних адміністративного агенту запит на свою авторизацію. У цьому запиті він також передає інформацію про свої власні потоки вводу-виводу. Це дозволяє агенту адміністрування СКУД додати цей агент контролю доступу до списку активних агентів контролю доступу системи та підключитися до потоків вводу виводу цього агенту контролю доступу. Після цієї операції, агенти можуть вільно взаємодіяти між собою використовуючи потоки вводу-виводу.

С технічної точки зору, протоколи взаємодії між агентами допускають роботу декількох адміністративних агентів у межах однієї СКУД, але така робота не вимагається від звичайної конфігурації системи. Таке рішення може знадобитися у разі необхідності мати резервний агент адміністрування СКУД на випадок що інший вийде з ладу, або втратить підключення до мульти-агентної системи.

3 АНАЛІЗ ТА ОБҐРУНТУВАННЯ РЕАЛІЗАЦІЇ АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Оцінка вимог

Дослідивши специфікацію різних СКУД, загальні вимоги до сучасних СКУД, та сучасних користувачів СКУД у порівнянні з конкурентними рішеннями було сформульовано список конкретних властивостей розроблюваної мульти-агентної системи[1, 2, 3, 4, 6, 7, 8, 19]:

- Мульти-агентна система повинна забезпечувати доступ авторизованим та неавторизованим особам відповідно до регламенту.
- Мульти-агентна система повинна відповідно реагувати на зовнішні та внутрішні чинники
- Мульти-агентна система повинна забезпечувати можливість взаємодії з іншими автоматизованими та неавтоматизованими зовнішніми системами
- Мульти-агентна система повинна мати можливість задавати різні рівні доступу до різних користувачів системи
- Мульти-агентна система повинна відповідати вимогам безпеки персоналу, що знаходиться у зонах під контролем системи
- Мульти-агентна система повинна бути захищена від несанкціонованого втручання
- Агенти системи повинні бути самостійними і продовжувати свій робочий цикл незважаючи на зовнішні обставини
- Кожен агент системи повинен мати надійний та захищений метод комунікації з іншими агентами мульти-агентної системи
- Кожен агент системи повинен мати відповідний інтерфейс обслуговування, що дозволить аналізувати його поточний стан
- Кожен агент системи повинен вести історію свого стану та історію подій

- Система повинна мати різні види агентів для виконання різних поставлених до неї задач
- Кожен інтелектуальний агент повинен бути гнучким до реконфігурації відповідно до мінливості вимог
- Кожен адміністративний агент системи повинен зберігати історію усіх агентів контролю доступу, з якими він співпрацює
- Кожен агент контролю доступу повинен мати корпус, мікрокомп'ютер, набір роз'ємів для підключення інших апаратних засобів, та модуль регуляції живлення
- Модуль управління живленням агенту контролю доступу повинен забезпечити можливість безпечно вмикати та вимикати пристрій, за необхідності

3.2 Вибір мікроконтролера для модуля управління живленням

Згідно вимог, модуль управління живленням повинен мати здатність вмикати та вимикати безпечно мікрокомп'ютер. Для цього, він повинен комунікувати з мікрокомп'ютером, відслідковувати стан кнопки ввімкнення/вимкнення та присутність живлення на стабілізаторі. Більшість доступних у вільному продажу мікроконтролерів успішно справляються з цією задачею а тому вибір буде визначатись за параметрами які не відносятся прямим чином до вимог.

На даний момент найбільш популярними архітектурами, що використовуються у мікроконтролерах, є ARM від виробників Renesas, ASMedia, Nuvoton, NXP, STMicroelectronics, Qualcomm, Texas Instruments та ін. а також AVR за виробництвом Microchip(Atmel). Для порівняння було обрано декілька моделей, спираючись на доступність у роздрібному продажі. Вибір архітектури контролеру не є принциповим, задачі в нього відносно прості.

Таблиця 3.1 – Порівняльна характеристика різних контролерів Atmega та STM32

Характеристика	Arduino Nano 328p	Arduino Pro Micro 32U4	STM32F103C8T6

Архітектура мікроконтролеру	Atmega AVR	Atmega AVR	ARM32 Cortex-M3
Частота роботи	16 МГц	16 МГц	72 МГц
Пам’ять програми	32 КБ	32 КБ	64 КБ
Оперативна пам’ять	2 КБ	2.5 КБ	20 КБ
Вид живлення	5В або 7-12В	3.3В або 6-12В	3.3В
RTC	Відсутній	Відсутній	Присутній
Кількість UART	3 шт.	3 шт.	Відсутній
Спосіб прошивки	USB	USB	Програматор STLink
Ціна	98 ₴	188 ₴	67 ₴

Хоча STM32 (рисунок 3.1) є має кращі характеристики при нижчій ціні, відсутність можливості його програмування через USB робить його не зручним для розробки.

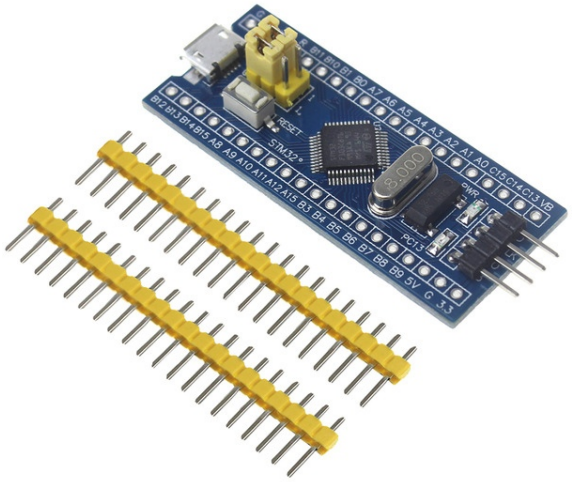

Рисунок 3.1 – Зображення апаратної платформи STM32F103C8T6

Контролер Atmega328P на базі апаратної платформи Arduino Nano (рисунок 3.2) хоч і значно слабший апаратно, має простий інтерфейс для його програмування.

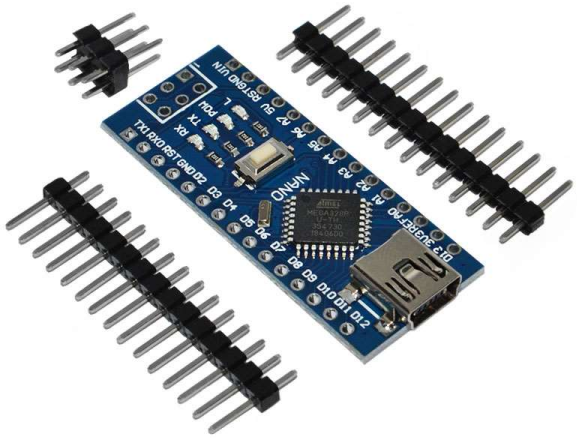

Рисунок 3.2 – Зображення апаратної платформи Arduino Nano 328p

Контролер Atmega32u4 на базі апаратної платформи Arduino Pro Micro (рисунок 3.3) у деяких аспектах краще ніж його молодша версія, але різниця у ціні двократна.

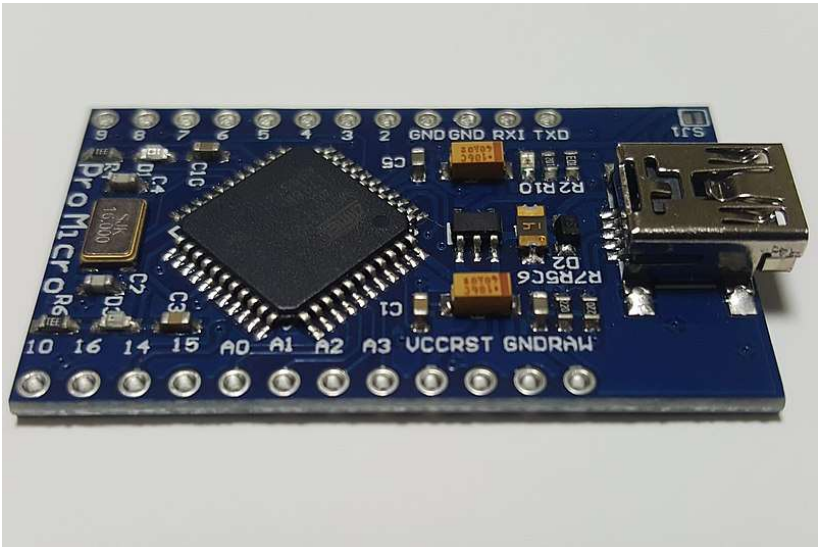

Рисунок 3.3 – Зображення апаратної платформи Arduino Pro Micro

Для виконання встановлених вимог, які дозволяють обирати мікроконтролер базуючись на простоті розробки та доступності ціни, було обрано мікроконтролер Atmega 328P. Для спрощення розробки апаратної платформи агенту контроля доступу, було прийнято рішення використовувати готову апаратну платформу мікроконтролера, а саме, Arduino Nano 328P.

Живлення плати Arduino Nano 328P може здійснюватися двома наступними способами:

- Використовуючи USB роз’єм на платі. Цей метод може використовуватись при програмування плати
- Використовуючи джерело живлення, під’єднане до відповідних контактів

Для апаратної платформи агенту управління доступом було обрано живити мікроконтролер використовуючи 5В живлення. Забезпечити таке живлення дозволяє стабілізатор постійного току XL4005.

Мікроконтролер має 14 цифрових виходів, які помічаються буквою D (digital). Контакти використовуються як входи та виходи, у кожного є підтягуючий резистор. Також мікроконтролер має аналогові входи, які позначаються літерою A (analog). Цифрові входи дозволяють мікроконтролеру зчитувати логічні стани, у той час як аналогові входи повертають значення напруги на вході. Аналогові входи позначені символом «~» можна використовувати у якості PWM входів.

Зокрема мікроконтролер має два виходи зовнішнього переривання, що дозволяє моментально реагувати на будь яку взаємодію з входами цього мікроконтролеру.

Обсяг пам'яті програм мікроконтролеру ATmega328 становить 32 КБ (з них 2 КБ використовуються загрузчиком. Крім цього, ATmega328 має 2 КБ оперативної пам'яті SRAM і 1 КБ EEPROM (для взаємодії з якої служить бібліотека EEPROM).

Загалом мікроконтролер має широкі функціональні можливості, проте має недоліки в дуже обмеженій оперативній пам’яті, низькій кількості виходів та функціональній обмеженності.

3.3 Програмування мікроконтролера для модуля управління живленням

Зазвичай, для роботи з цією апаратною платформою рекомендують використовувати середовище програмування Arduino IDE. Але після ознайомлення з його функціоналом, було прийнято рішення користуватись більш функціональним та

професійним середовищем програмування. Середовище програмування грає ключову роль у простоті розробки програмного забезпечення за визначеними вимогами.

Від середовища програмування вимагаються наступні властивості:

- Автодоповнення введених команд
- Аналіз коду та пошук помилок
- Розширений функціонал редагування тексту
- Можливість вибору компілятора
- Вбудований термінал з функціоналом UART протоколу для взаємодії з апаратною платформою
- Можливість інтеграції з засобами контролю версій програми
- Простота доповнення проекту сторонніми бібліотеками

До вимог, що описані вище, наближаються тільки два програмних комплекси: Eclipse IDE та JetBrains CLion.

Таблиця 3.2 – Порівняльна характеристика середовищ розробки програмного забезпечення JetBrains Clion та Eclipse IDE.

Властивість	Eclipse IDE	JetBrains Clion
Автодоповнення введених команд	Так	Так
Аналіз коду та пошук помилок	Так	Так
Розширений функціонал редагування тексту	Так, але не ідеальний	Так
Можливість вибору компілятора	Так, присутні заготовлені шаблони	Так
UART термінал	Так, але має недоліки	Ні/не працює
Можливість інтеграції з засобами контролю версій програми	Так, але інтерфейс незручний	Так
Простота доповнення проекту сторонніми бібліотеками	Так, присутній репозиторій бібліотек	Так

Eclipse IDE має широкий набір функціоналу та забезпечує розробника необхідним мінімумом автоматизованого функціоналу. Його головним недоліком можна назвати загальну застарілість інтерфейсу користувача, та підходів його роботи.

JetBrains Clion є адаптацією популярної JetBrains Intellij IDEA для програмування на мовах C/C++. Це середовище програмування наслідує багато переваг та властивостей свого нащадку, але сирість деяких інструментів розробки є значним недоліком.

За результатом порівняння схожих середовищ програмування, для розробки програмного забезпечення модуля управління живленням було обрано Eclipse IDE.

Мікроконтролери ATmega можуть бути запрограмовані за допомогою мов програмування C та C++. Eclipse IDE підтримує мови C і C++, використовуючи стандартні правила структуризації коду, та доповнюючи їх специфічними правилами структуризації коду для AVR архітектури. Eclipse IDE постачає бібліотеку програмного забезпечення від проекту Wiring, яка забезпечує безліч загальних процедур введення та виведення даних. Написаний користувачем код вимагає лише двох основних функцій для запуску скетчу (програмного застосунку) та основного циклу програми, які компілюються та пов'язуються із заглушкою функції main() у виконувану циклічну програму. У Eclipse IDE використовується консольне програмне забезпечення avrdude для компіляції та завантаження програмного забезпечення на апаратну платформу Arduino.

3.4 Вибір методу комунікації між агентами мульти-агентної системи

Для забезпечення надійної роботи системи, необхідно забезпечити достатній рівень комунікації окремих агентів між собою. Для спрощення процесу розробки програмного забезпечення, було обрано використовувати брокер повідомлень. Обраний брокер повідомлень має забезпечувати безпечну та надійну передачу повідомлень між агентами. Крім цього, він повинен бути кроссплатформенним, що

дозволить використовувати такий метод комунікації з різними апаратними платформами.

Брокер повідомлень – це програмний засіб, який перетворює повідомлення з протоколу обміну повідомленнями відправника в протокол обміну повідомленнями приймача. Брокери повідомлень – це елементи в телекомунікаційних або комп'ютерних мережах, де програмні програми спілкуються, обмінюючись формально визначеними повідомленнями[20].

Брокер повідомлень – це архітектурна схема для перевірки, трансформації та маршрутизації повідомлень. Він забезпечує комунікацію між додатками, зводячи до мінімуму взаємну обізнаність про взаємодіючі програмні рішення. [21]

Основна мета брокера – приймати вхідні повідомлення з програм та виконувати деякі дії над ними. Брокери повідомлень можуть роз'єднувати кінцеві точки, відповідати конкретним нефункціональним вимогам та полегшити повторне використання функцій посередників. Наприклад, брокер повідомлень може використовуватися для управління чергою навантаження або чергою повідомлень для декількох приймачів, забезпечуючи надійне зберігання, гарантовану доставку повідомлень і, можливо, управління транзакціями. Далі представлені інші приклади функцій, які може виконувати брокер [19, 20]:

- Спрямовувати повідомлення до одного або декількох напрямків;
- Перетворювати повідомлення в альтернативне подання (альтернативна реалізація);
- Виконувати агрегацію повідомлень, розклавши повідомлення на кілька повідомлень та надсилаючи їх до місця призначення, після чого перекомпонувавши частини в одне повідомлення для повернення програмі-користувачу;
- Взаємодіяти із зовнішнім сховищем, для доповнення повідомлення або його збереження;
- Завантажити веб-службу для опрацювання даних;
- Реагувати на події чи помилки при комунікації програмних засобів;

- Надавати маршрутизацію повідомлень на основі їх вмісту та теми, використовуючи шаблон публікації та підписки;

Брокери повідомлень, як правило, базуються на одній із двох фундаментальних архітектур: hub-and-speak та bus message. По-перше, центральний сервер виступає як механізм, який надає послуги інтеграції, тоді як з останнім, брокер повідомлень є магістральним каналом зв'язку або розподіленою службою, яка діє на шині. [20] Крім того, для інтеграції декількох посередників може бути використаний більш масштабований підхід з декількома брокерами. [20]

Для мульти-агентної системи контролю та управління доступом було вирішено обрати брокер повідомлень Eclipse Mosquitto MQTT.

Eclipse Mosquitto - це брокер повідомлень з відкритим кодом, який реалізує протокол MQTT. Mosquitto є легковажним брокером, що підходить для використання на всіх пристроях, від планшетних комп'ютерів низької потужності до повноцінних серверів.

Протокол MQTT забезпечує простий метод здійснення обміну повідомленнями за допомогою моделі публікації / підписки. Це робить його придатним для обміну повідомленнями у IoT системах, таких як датчики низької потужності або мобільні пристрої, такі як телефони, вбудовані комп'ютери або мікроконтролери.

Для забезпечення надійного розбору отриманих даних, дані потрібно серіалізувати та десеріалізувати під час передачі та отримання повідомлень від брокеру повідомлень.

3.5 Вибір мікрокомп’ютеру для агенту контролю доступу

Для забезпечення комунікації між агентами, було вирішено використовувати протокол передачі даних MQTT. Для збору даних обробки та забезпечення виконання задач відповідно до регламенту, потрібно було обрати плату, яка відповідала б вимогам апаратно-програмного комплексу агенту контролю доступу.

Ринок мікрокомп’ютерів має багато представників та багато моделей. Після проведеного дослідження, було визначено декілька найбільш перспективних

моделей, що мають необхідні властивості: RaspberryPi 3, RaspberryPi 4, OrangePi PC, OrangePi One (рисунок 3.5) та OrangePi 4. Перелічені мікрокомп’ютери є моделями двох відомих виробників мікрокомп’ютерів, які забезпечують достатній рівень підтримки програмного забезпечення своїх систем та чудовим показникам у залежності ціни-якості.

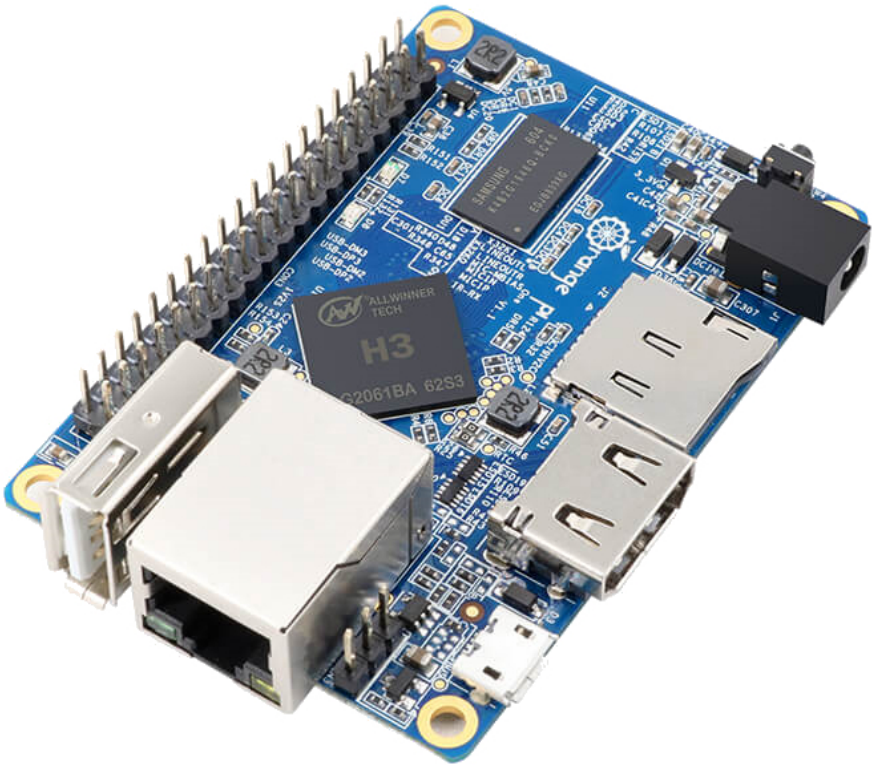

Рисунок 3.5 – Мікрокомп’ютер OrangePi One

В першу чергу, між різними пристроями вирізняється параметр вартості. OrangePi PC та One є дуже доступними апаратними платформами, коштуючи значно дешевше, ніж RaspberryPi 3, 4 та OrangePi 4. Це пов’язано з тим, що в OrangePi PC та One використовується дешевий старий процесор китайського виробництва. Незважаючи на ціну, процесор має достатню потужність для виконання задач агенту контролю доступу.

На вартість апаратної платформи значно впливає бренд. RaspberryPi є англійською компанією, що започаткувала напрямок мікрокомп’ютерів, тому і ціна подібних виробів значно вища. При виникненні проблем з китайським аналогом,

вирішувати їх доведеться самостійно, в той час як RaspberryPi має велике товариство користувачів, у яких можна спитати будь-що.

Іншим важливим фактором є вибір підтримуваних операційних систем. Апаратні платформи OrangePi підтримують велику кількість ОС. На офіційному сайті виробника можна переглянути список і звірити, які системи підходять для тієї чи іншої версії плати. На жаль, повна підтримка доступна лиша на одній-двох ОС. У цей же час, RaspberryPi має широкий вибір підтримки операційних систем з повністю робочим функціоналом.

Після проведеного детального аналізу апаратних засобів, доступного програмного забезпечення, вибору доступних для встановлення операційних систем з повною підтримкою роботи апаратних засобів, вибір зменшився до двох моделей (таблиця 3.3): OrangePi PC та One.

Таблиця 3.3 – Порівняльна характеристика мікрокомп’ютерів OrangePi PC та One

Характеристика	OrangePi PC	OrangePi One
Процесор	Allwinner H3	Allwinner H3
Частота роботи	1,2 ГГц	1,2 ГГц
Об’єм оперативної пам’яті	1 ГБ	512 МБ
Кількість USB роз’ємів	3	1
Роз’єм RJ-45	Присутній	Присутній
Живлення	5В 2А через роз’єм або GPIO	5В 2А через роз’єм або GPIO
Пам’ять пристрою	microSD слот	microSD слот
Вартість	655₴	495₴

Фактично, ці моделі майже однакові за своєю суттю. Вибір складається у необхідності доплати 25% вартості за деякі спрощення при розробці та використанні. У результаті, для розробки агенту контролю доступу було обрано мікрокомп’ютер OrangePi PC.

Orange Pi PC (рисунок 3.6) – Linux-сумісний мікрокомп’ютер на архітектурі ARM. На ньому можна використовувати такі операційні системи як Android, Ubuntu та її варіації, Debian, Raspbian, Armbian, та інші. Ця апаратна платформа обладнана процесором AllWinner H3, що включає в собі 4 ядра на архітектурі Cortex A7, і обладнана 1 ГБ оперативної пам’яті[10].

Рисунок 3.6 – Мікрокомп’ютер Orange Pi PC

Обрана модель мікрокомп’ютеру є дуже гнучкою у застосуванні (рисунок 3.6).

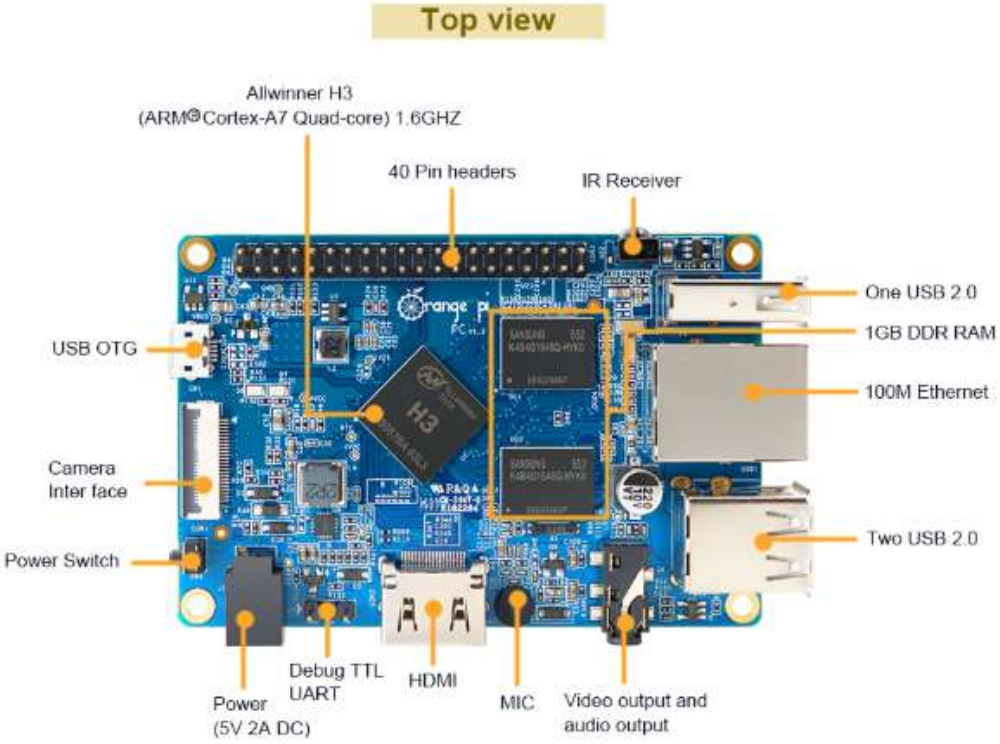

Рисунок 3.6 – Структура мікрокомп’ютеру OrangePi PC

Використовуючи 4 UART, 40 контактів GPIO та Ethernet-модуль цей мікрокомп'ютер можна перетворити на будь-що, використовуючи відповідні програмні та апаратні засоби. Такий мікрокомп’ютер є дуже дешевим функціональним рішенням для прототипування або розробки програмно-апаратних комплексів будь-якої складності[10].

Найважливішою перевагою цієї апаратної платформи є можливість встановлення повноцінної операційної системи. Операційна система завантажується на microSD карту, що вставляється у відповідний роз’єм[10].

Дуже важливим параметром для спрощення розробки програмного забезпечення (а іноді, взагалі, виступає перепоною для подальшої розробки) – це оперативна пам’ять комп’ютера. Саме ця модель, на відміну від One, обладнана оперативною пам’яттю об’ємом 1 ГБ. Такий об’єм дозволяє розробляти програмне забезпечення не зважаючи на оптимізацію використання оперативної пам’яті[10].

Виробник OrangePi рекомендує використовувати блоки живлення постійного струму з напругою 5 Вольт і силою струму 2 Ампер. Це повинно виключити проблеми з запуском пристроїв і гарантує стабільну роботу приладу. Але на практиці, апаратна платформа агенту контролю доступу споживає не більше 1 А під час роботи[10].

3.6 Вибір операційної системи мікрокомп’ютера агенту контролю доступу

Для своєї роботи, апаратна платформа мікрокомп’ютеру вимагає встановлення операційної системи. Обрана нами апаратна платформа OrangePi PC підтримує широкий вибір операційних систем на базі Linux: Ubuntu, Raspbian, Debian, Armbian, Android та інші.

Серед широкого переліку операційних систем, лише дві мають повноцінну підтримку всіх апаратних засобів обраної апаратної платформи – Ubuntu та Armbian.

При подальшому порівнянні, вибір операційної системи було зроблено на користь Armbian,

Armbian Linux є оптимізованими образами Debian і Ubuntu Linux для мікрокомп’ютерів на базі ARM. У цієї операційної системи існує велика екосистема різних апаратних платформ, які виступають потужною альтернативою RaspberryPi. Головна задача Armbian полягає у створенні єдиної системи, що є надійною для роботи на будь-якому з десятків мікрокомп'ютерів ARM[17].

Основні переваги операційної системи Armbian[10, 17]:

- Звичайна Bash оболонка, стандартні утиліти Debian / Ubuntu. Загальні та специфічні налаштування можно проводити через простий програмний застосунок з інтерфейсом користувача. Користувач може підключатись до мікрокомп’ютеру через серійний порт, HDMI / VGA або SSH.
- Встановлюється без надлишкового програмного забезпечення чи шпигунських програм. Спеціальні утиліти абсолютно необов’язкові.
- Розподілений на розділи дистрибутив ущільняється до реального розміру даних і вміщується приблизно в 1G. Дистрибутив оптимізовано для використання з SD-карти. Встановлення програм суттєво скорочує термін служби SD-карти. SD-карти не були розроблені для такого типу використання.
- Мікрокомп’ютери оптимізовані на рівні ядра та на рівні роботи користувачів. Оптимізація DVFS, кешування журналу пам’яті, кешування пам’яті профілю браузера, налаштування користування файлом підкачки, затримка очищення сміття. Ця система працює майже повністю у режимі тільки читання і є однією з найшвидших версій Linux для багатьох мікрокомп’ютерів.
- Рівень безпеки системи знаходиться на рівні Debian / Ubuntu і може бути посилений за допомогою додаткової конфігурації. Це хороша відправна точка для промислового та побутового використання. Систему регулярно перевіряють професіонали серед користувачів цієї системи. Кожна офіційна стабільна збірка ретельно перевіряється.

- Забезпечення довгострокових оновлень, виправлень безпеки, документації, підтримки користувачів.
- Збірник системи та ядра з відкритим кодом, розробка, обслуговування та розповсюдження більш ніж 30 різних ядер ARM та ARM64 Linux. Потужні засоби побудови та розробки програмного забезпечення. Може працювати в повністю паралельному режимі. Може працювати під Docker.

Для роботи з апаратними засобами GPIO на платформі OrangePi PC, необхідно додатково встановити програмний засіб WiringOP. Цей програмний засіб є копією WiringPi та виконує функції API до GPIO апаратної платформи[11].

4 ЗАСОБИ РОЗРОБКИ

Увесь процес розробки програмно-апаратного комплексу СКУД можна розділити на декілька основних частин: розробка програмного комплексу агенту контролю доступу, розробка програмного комплексу агенту адміністрування СКУД, розробка апаратної платформи агенту контролю доступу.

Програмне забезпечення контролеру Atmega 328P, що є частиною апаратної платформи агенту контролю доступу, було розроблено мовою C++. Для розробки програмного застосунку було використано Eclipse IDE, з пакетом збірки для цього контролеру.

Апаратна платформа агенту контролю доступу розроблялася за допомогою програмного забезпечення Autodesk EAGLE. На основі результату розробки у CAD було створено, лабораторією кібер-енергетичних систем, апаратну платформу, що вмістила у собі всі компоненти агенту контролю доступу.

На мікрокомп’ютер, що є частиною апаратної платформи агенту контролю доступу, було встановлено операційну систему Armbian, програмний пакет WiringOP та середовище виконання Java Runtime Environment.

Програмне забезпечення агенту контролю доступу створювалось на мові програмування Java. У якості СУБД була обрана вбудована БД H2, а для роботи з нею використовувався Hibernate ORM. Взаємодія з апаратною платформою виконувалось з використанням бібліотеки WiringOP.

Програмне забезпечення агенту адміністрування СКУД було створено мовою програмування Java, з використанням фреймворку Spring Boot. У якості СУБД була обрана MySQL, а для роботи з нею використовувався Spring Data. Для розмежування рівнів доступу користувачів використовувалася система захисту та авторизації Spring Security. Інтерфейс взаємодії адміністратора з агентом адміністрування СКУД було інтегровано у середовище виконання інтелектуального агенту використовуючи Vaadin Framework. Для взаємодії між собою агенти використовують брокер повідомлень Eclipse Moquitto MQTT.

Для розробки програмного забезпечення мовою Java використовувалось середовище розробки JetBrains Intellij IDEA. Для розробки усіх баз даних використовувалось програмне забезпечення JetBrains Datagrip.

4.1 Програмні засоби та інструменти для розробки програмно апаратного комплексу

Розробка апаратної платформи – це складний процес створення не тільки програмного забезпечення, але і фізичної моделі кінцевого пристрою. Для значного спрощення процесу, при моделюванні використовувалось CAD програмне забезпечення Autodesk EAGLE.

Для розробки програмного забезпечення мовою Java було використано середовище програмування JetBrains Intellij IDEA.

IntelliJ IDEA - інтегроване середовище розробки програмного забезпечення для багатьох мов програмування, зокрема Java, JavaScript, Python, розроблена компанією JetBrains[18].

Перша версія з'явилася в січні 2001 року і швидко набула популярності як перше середовище для розробки мовою Java з широким набором інтегрованих інструментів для рефакторингу, які дозволяли програмістам швидко реорганізовувати вихідні тексти програм. Дизайн середовища орієнтований на продуктивність роботи програмістів, дозволяючи сконцентруватися на функціональних завданнях, в той час як IntelliJ IDEA бере на себе виконання рутинних операцій[18].

IntelliJ IDEA надає інтегрований інструментарій для розробки графічного інтерфейсу користувача. Серед інших можливостей, середовище розробки сумісне з багатьма популярними вільними інструментами розробників, такими як Git, Subversion, Apache Ant, Maven і JUnit[18].

Середовище розробки доступне в двох редакціях: Community Edition і Ultimate Edition. Community Edition є повністю безкоштовною версією, доступною під ліцензією Apache 2.0, в ній реалізована повна підтримка мов програмування Java SE, Kotlin, Groovy, Scala, а також інтеграція з найбільш популярними системами

управління версіями. В редакції Ultimate Edition, доступною під комерційною ліцензією, реалізована підтримка Java EE, UML-діаграм, підрахунок покриття коду тестами, а також підтримка інших систем управління версіями, мов та фреймворків[18].

4.2 Програмування агенту контролю доступу

Комбінація обраної операційної системи та програмної платформи дозволяє обирати мову програмування з великої кількості різних варіантів. Але, зважаючи на жорсткі вимоги до роботи цього елементу системи, для розробки була обрана мова програмування Java.

Один з основних елементів програмного забезпечення агенту контролю доступу є СУБД. В умовах жорсткої економії оперативної пам’яті, для виконання задач СУБД агенту контролю доступу було обрано H2 у режимі роботи embedded.

H2 Database – Java SQL БД. Ця СУБД має наступні особливості[15]:

- Дуже швидка СУБД з відкритим кодом, що використовує JDBC API;
- Декілька режимів роботи: вбудований та серверний режими; БД у RAM;
- Веб-застосунок для адміністрування;
- Маленький розмір: файл jar в розмірі близько 2 МБ.

Обрана мова програмування вимагає використання відповідного JDBC драйверу для підключення до будь-якої СУБД. Для спрощення процесу розробки, та підвищення якості програмного продукту, підключення до СУБД виконується з використанням Hibernate ORM.

Hibernate ORM – найпопулярніша реалізація специфікацій JPA та JDBC, призначена для вирішення завдань об'єктно-реляційного відображення (ORM)[24].

JPA (Java Persistence API) є специфікацією Java API для управління стійкістю (persistence) та об'єктом / реляційним відображенням з Java EE та Java SE. Технічним завданням цієї роботи є надання об'єкта / реляційного картографування для розробника Java-програм за допомогою Java моделі домену для управління реляційною базою даних[22].

Сутність це легкий зберігаємий об'єкт бізнес логіки (persistent domain object). Основна програмна сутність – це клас, який так само може використовувати додаткові класи, який можуть використовуватися як допоміжні класи або для збереження стану сутності[23].

ORM означає об'єктно-реляційне відображення (ORM) - це програма для перетворення даних між реляційними базами даних та об'єктно-орієнтованими мовами програмування, такими як Java, C # тощо[24].

Система ORM має наступні переваги перед звичайним JDBC[25]:

- Надає об'єкти доступу до бізнес-коду, замість таблиці БД.
- Приховує деталі SQL-запитів з ОО-логіки.
- Базується на JDBC "під капотом".
- Не потрібно мати справу з впровадженням бази даних.
- Суб'єкти, засновані на бізнес-концепціях, а не на структурі бази даних.
- Управління транзакціями та автоматичне створення ключів.
- Швидкий процес розробки програмного забезпечення.

4.3 Програмування адміністративного агенту СКУД

Завдяки загальній гнучкості архітектури системи, адміністративний агент СКУД може приймати будь-яку форму, що задовольняє вимоги агенту. Цей агент може бути хмарним програмним забезпеченням, може бути встановленим на один з агентів контролю доступу тощо. Вибір мови програмування також вільний, як і СУБД.

Для спрощення процесу розробки було обрано мову програмування Java, аналогічно до агенту контролю доступу. Для спрощення процесу розробки, тестування та адаптації до різних середовищ виконання, було прийнято рішення використовувати Spring Boot Framework.

Spring Framework пропонує повноцінну модель програмування та конфігурації сучасних промислових Java-програм на будь-якій платформі розгортання.

Основні переваги Spring Framework[14]:

- Основні технології: dependency injection, події, файлові ресурси, інтернаціоналізація, валідація, прив'язка даних, перетворення типів, SpEL, AOP.
- Тестування: mock об’єктів, фреймворк тестування Spring MVC Test.
- Доступ до даних: транзакції, підтримка DAO, JDBC, ORM, Marshalling XML.
- Веб-фреймворки Spring MVC та Spring WebFlux.
- Інтеграція: віддалена взаємодія, JMS, JCA, JMX, електронне листування, створення завдань, планування виконання роботи, кешування даних.

Ключовим елементом Spring є інфраструктурна підтримка на рівні прикладних програм: Spring фокусується на «сантехніці» корпоративних додатків, щоб команди могли зосередитись на реалізації бізнес-логіки на рівні додатків без зайвої взаємодії із конкретними середовищами розгортання[14].

Для роботи з СУБД адміністративного агенту СКУД було обрано ORM Spring Data, що входить до складу Spring Boot Framework та базується на Hibernate ORM та стандарті JPA[14, 24, 22].

Основна задача Spring Data полягає у наданні звичної та постійної, Spring моделі програмування доступу до даних, зберігаючи при цьому особливості використовуваного сховища даних. Це полегшує використання технологій доступу до даних, реляційних та нереляційних баз даних, та хмарних служб передачі даних. Це комплексний проект, який містить багато малих проектів, характерних для певної бази даних. Проекти розробляються спільно з багатьма компаніями та розробниками, які стоять за цими технологіями[14].

Серед переваг Spring Data можна зазначити[14]:

- Потужна реалізація паттерну repository та спеціальні абстракції для відображення об'єктів
- Динамічне створення запитів на базі імен методів, що визначені у паттерні repository
- Реалізація базових класів доменів, що забезпечують основні властивості

- Підтримка прозорого аудиту змін у БД
- Можливість створення власної реалізації паттерну repository
- Проста Spring інтеграція через JavaConfig та користувацькі XML
- Розширена інтеграція з контролерами Spring MVC

Для забезпечення високого рівня безпеки системи та її взаємодії з адміністратором було вирішено всю зовнішню взаємодію з системою обгорнути системою авторизації на базі Spring Security.

Найважливішими перевагами Spring Security виступають[14]:

- Повна та розширювана підтримка як для автентифікації, так і для авторизації
- Захист від атак, таких як встановлення сеансу, підробка веб-сайтів, тощо
- Інтеграція у Java Servlet API
- Можлива інтеграція з Spring Web MVC, Vaadin Framework та іншими веб-системами

Також, для спрощення процесу розробки інтерфейсу адміністратора, було прийнято рішення інтегрувати його у середовище виконання агенту адміністрування СКУД, використовуючи Vaadin Framework.

Vaadin дозволяє легко створювати красиві веб-додатки на Java. Включена бібліотека компонентів інтерфейсу призначена для роботи як на мобільному, так і на повноформатному екрані. Користувачі оцінять увагу до деталей, тоді як розробник можете зосередитись на створенні функціональності[13].

4.4 Мова програмування Java

Java – об'єктно-орієнтована мова програмування. Серед основних переваг – висока надійність та відмовостійкість, висока швидкість виконання, широка сумісність з різними платформами та операційними системами, та повна кросплатформенність програм, написаних цією мовою програмування[18].

Розробка програмного забезпечення мовою програмування Java базується на принципах об’єктно-орієнтованого програмування.

ООП об’єктно-орієнтоване програмування – є парадигмою програмування, яка описує рішення задачі програмування як моделювання сукупності об'єктів, що співпрацюють між собою та розділюються за функціональними ознаками.

Об'єкт – це сутність, яка має свої властивості, атрибути, поведінку і є екземпляром відповідного класу. Внутрішні дані об’єкту, як правило, приховані від інших об'єктів, з чого випливає що єдиний спосіб взаємодіяти з даними об’єкту через методи самого об'єкту.

Клас – це водночас шаблон для створення об’єктів та спеціальний статичний вид об’єкту. Клас має властивість успадковування атрибутів і логіки роботи інших класів (батьківських), і може описувати свою реалізацію їх методів.

Описані характеристики класів та об’єктів можна охарактеризувати наступними фундаментальними концепціями ООП:

- Інкапсуляція – це концепція ООП, яка поєднує в між собою дані та функції, що управляють даними, і захищає ці дані від зовнішньої взаємодії та неправильного використання. Інкапсуляція даних призвела до важливої концепції ООП приховування даних.
- Наслідування – це механізм базування об'єкта або класу на іншому об'єкті (наслідування на основі прототипу) або класі (наслідування на основі класу), зберігаючи аналогічну реалізацію. Також визначається як отримання нових класів (підкласів) із існуючих, таких як базовий клас, а потім їх формування в ієрархію класів.
- Поліморфізм виділяється наступними поняттями:
    - Перевантаження – це різновид поліморфізму, при якому поліморфні функції можуть бути застосовані до аргументів різних типів, оскільки поліморфна функція може позначати ряд чітких та потенційно гетерогенних реалізацій залежно від типу аргументів (аргументів), яким вона є застосовується.
    - Узагальнюючий поліморфізм – це спосіб зробити мову більш виразною, зберігаючи при цьому повну безпеки типізації.

Використовуючи такий вид поліморфізму, функцію або тип даних можна записати узагальнено, щоб вона могла обробляти значення вхідних даних однаково, не залежно від їх типу.

- Перевизначення – це форма поліморфізму типу, в якій підтип - це тип даних, що пов'язаний з іншим типом даних (супертипом) деяким поняттям замінюваності, тобто програмні елементи, як правило, підпрограми або функції, записані на оперувати елементами супертипу можуть також оперувати елементами підтипу.

Для забезпечення кросплатформенності, та виконання слогану WORA (Write Once Run Anywhere) мова програмування Java використовує JVM – Java Virtual Machine[18].

Java Virtual Machine – це віртуальна машина, яка дозволяє комп'ютеру запускати програми Java, а також програми, написані іншими мовами, які також компілюються в Java байт-код. JVM деталізується специфікацією, яка формально описує, що потрібно для реалізації JVM. Наявність специфікації забезпечує сумісність програм Java у різних реалізаціях, так що авторам програм, що використовують комплект Java Development Kit (JDK), не потрібно турбуватися про характерні риси базової апаратної платформи[18].

5 ВИКОРИСТАННЯ СИСТЕМИ

Для забезпечення правильної роботи системи та виконання вимог СКУД, вона має бути правильно встановлена та сконфігурована відповідним спеціалістом. Адміністратор системи може впливати на систему, використовуючи веб-застосунок.

5.1 Запуск роботи агенту контролю доступу

Для початку роботи мульти-агентної системи, вона повинна включати хоча б один встановлений програмно-апаратний комплекс агенту контролю доступу (рисунок 5.1)

Рисунок 5.1. Фото модулю контролю доступу з сторони органів управління.

Агент контролю доступу вимагає підключення до себе наступних складових:

- джерела живлення 12В;
- модулю авторизації користувача
- доступу до локальної мережі

- модулю забезпечення доступу

Підключення джерела живлення, модулю забезпечення доступу та доступу до локальної мережі зображено на рисунку 5.1.

Підключення модулю авторизації користувача зображено на рисунку 5.2. Також, на цьому рисунку зображено вільні роз’єми, що дозволяють використовувати водночас декілька модулів авторизації користувача.

Рисунок 5.2. Фото модулю контролю доступу з сторони роз’ємів підключення модулів авторизації користувача.

Після підключення всіх апаратних засобів, модуль готовий до ввімкнення. Для ввімкнення, необхідно натиснути на кнопку ввімкнення/вимкнення. Після цього, почнеться процес ініціалізації пристрою. Якщо прилад налаштовано правильно, зелений світлодіод на передній панелі почне світитися. Якщо процедура виконана неправильно, загориться червоний або жовтий світлодіод (рисунок 5.3).

Рисунок 5.3. Фото модулю контролю доступу з сторони органів управління. Жовтий світлодіод позначає несправність конфігурування програмного забезпечення.

Для роботи, модуль повинен мати доступ до загального (для СКУД) брокеру повідомлень Moquitto, що може знаходитись як у локальній мережі, так і у мережі Інтернет. У разі відсутності доступу до нього, або неправильної конфігурації, загориться жовтий світлодіод.

Якщо апаратне забезпечення модулю вийшло з ладу, або конфігурація апаратного забезпечення порушена, загориться червоний світлодіод.

5.2 Запуск адміністративного агенту СКУД

Для початку роботи, адміністративний агент СКУД повинен бути у будь-якому сумісному середовищі виконання та мати доступ до загального брокеру повідомлень

системи. Після ініціалізації, адміністратору стане доступним вікно авторизації (рисунок 5.4).

Рисунок 5.4. Форма авторизації інтерфейсу адміністратора.

Після авторизації, адміністратор може почати користуватись системою. Йому будуть доступні (рисунок 5.4) опції редагування регламенту в межах вже сконфігурованих модулів контролю доступу.

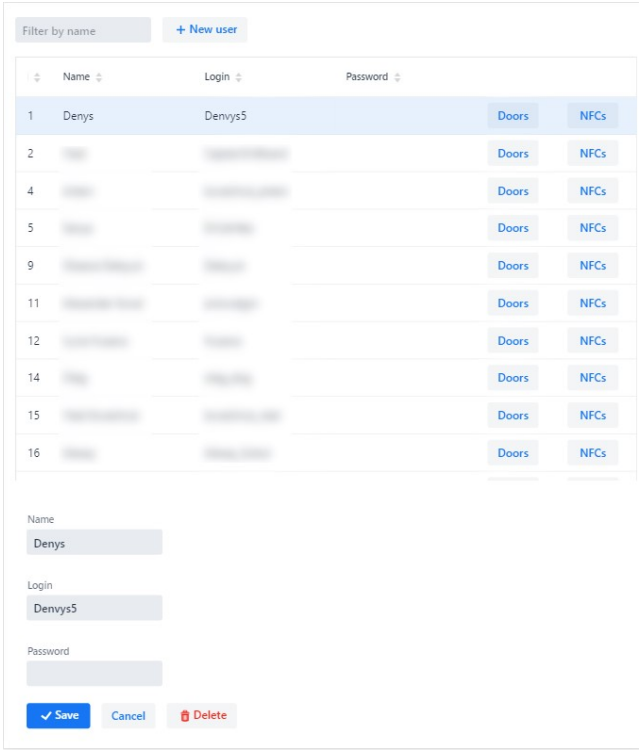

Рисунок 5.4. Інтерфейс редагування регламенту роботи СКУД.

У цьому вікні адміністратор може переглядати список існуючих користувачів. Натиснувши на поле користувача, з’являється інтерфейс для редагування інформації про користувача. Натискання кнопки «New user» викликає інтерфейс створення нового користувача системи.

Якщо адміністратор натисне на кнопку «Doors» то він отримає доступ до внесення змін у регламент відносно переліку приміщень та територій, що стосуються конкретного користувача (рисунок 5.5)

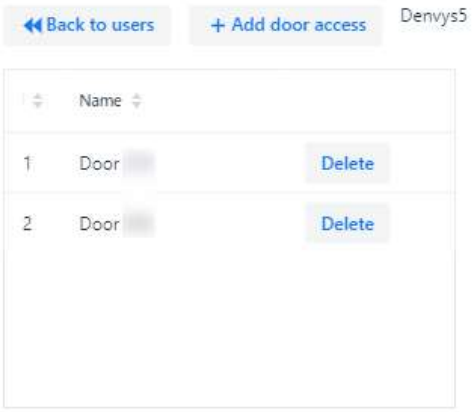

Рисунок 5.5. Інтерфейс редагування регламенту роботи СКУД для визначеного користувача системи.

Цей інтерфейс дозволяє у реальному часі редагувати доступ користувача до зазначених у переліку приміщень, або територій. Натискання на кнопку «Delete» забирає доступ у користувача, а кнопка «Add» надає користувачу доступ.

Якщо адміністратор натисне на кнопку «NFCs» то він отримає доступ до внесення змін у регламент відносно переліку методів авторизації конкретного користувача (рисунок 5.6).

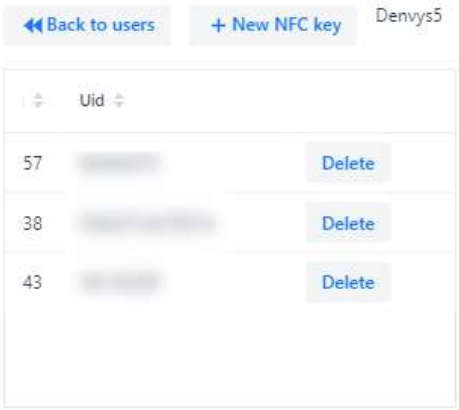

Рисунок 5.6. Інтерфейс редагування методів авторизації визначеного користувача системи.

Цей інтерфейс дозволяє у реальному часі редагувати перелік методів авторизації користувача. Натискання на кнопку «Delete» видаляє метод авторизації користувача,

а кнопка «New key» відкриває інтерфейс створення нового методу авторизації цього користувача.

5.3 Користування системою контролю та управління доступом

Після успішного встановлення та налаштування мульти-агентної системи, вона може бути офіційно прийнята у користування.

Для того щоб отримати доступ до бажаного приміщення, чи території, типовий користувач системи повинен підійти до розмежувача територій (наприклад двері), та використати свій метод авторизації (наприклад, прикласти NFC картку до NFC зчитувачу, що зображений на рисунку 5.7).

Рисунок 5.7. NFC-зчитувач у пластиковому корпусі.

Якщо даний користувач має доступ до приміщення, або території, що знаходиться по іншу сторону від розмежувача, то система надасть йому доступ.

ВИСНОВКИ

У ході аналізу існуючих рішень в області систем контролю та управління доступом, було досліджено розподілені та типові системи. Аналіз показав, що сучасні системи мають значні недоліки та малий простір для конфігурації та адаптації.

Розроблений програмно-апаратний комплекс є актуальним, оскільки розроблявся як конкурент існуючих систем, що, завдяки іноваційному підходу, облишений їх недоліків. Створену мульти-агентну систему можна використовувати як самостійну СКУД, або інтегрувати разом з іншими системами моніторингу та управління.

Було проведено огляд методів і засобів розробки програмного та апаратного забезпечення. Обґрунтовано вибір підходів до створення окремих елементів системи, а також вибір допоміжних програмних засобів. Це дало змогу створити гнучку систему, з можливостями легко адаптуватись під поточні вимоги.

Результати проведеного тестування роботи програмно-апаратного комплексу у межах лабораторії кібер-енергетичних систем підтверджують коректність роботи мульти-агентної системи, що показує повноту виконання поставлених вимог.

Ця розробка має найбільший потенціал у об'єктах з високим рівнем безпеки та складною ієрархією доступу персоналу до приміщень чи територій. Після консультацій військові підрозділи та великі бізнес-компанії зацікавилися цією розробкою.

Впровадження такої системи контролю дозволить підприємствам з високим рівнем внутрішньої безпеки збільшити економію внутрішніх ресурсів (контролюючи кількість людей у різних зонах та приміщеннях будівель), покращити якість звітності (про рух авторизованого персоналу всередині об'єктів, що контролюються системою) та зменшити обсяг накладних витрат на обслуговування систем контролю та управління доступом.

Робота над дипломним проектом покращила обізнаність у різноманітних технологій, що використовуються під час розробки програмного та апаратного забезпечення. Крім цього, було досліджено різноманітні техніки та алгоритми роботи

систем контролю та моніторингу, а також було досліджено специфіку сфери охорони та безпеки, для яких можна застосовувати нові навички. Також було створено робочий продукт програмно-апаратного комплексу, що виконує поставлені на нього задачі у межах лабораторії кібер-енергетичних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wooldridge M.J. An introduction to multi-agent systems. Wiley, 1996.
2. Тарасов В.Б. От многоагентных систем к интеллектуальным организациям. Философия, психология, информатика. М., Эдиториал. 2002
3. Ghallab M., Nau D., Traverso P. Automated planning: Theory & Practice. Morgan Kaufmann, 2004
4. Ворона В. А., Тихонов В. А. В83 Системы контроля и управления доступом. - М.: Горячая линия Телеком, 2010. - 272
5. Bellifemine F, Caire G, Greenwood D (2007) Developing multi-agent systems with JADE. Wiley, London
6. D. Dimarogonas, E. Frazzoli, K. Johansson, "Distributed event-triggered control for multi-agent systems", IEEE Trans. Autom. Control, vol. 57, no. 5, 2012.
7. Benantar, Messaoud. (2006). Access control systems. Security, identity management and trust models. Access Control Systems: Security, Identity Management and Trust Models. 10.1007/0-387-27716-1.
8. Vysoven D. “Intelligent agent of access management and control system”, International Competition of Student Scientific Works Black Sea Science 2020, Odessa, ONAFT 2020
9. Orange Pi. Wiring Op library. [Електронний ресурс]. – Режим доступу: http://www.orangepi.org/Docs/WiringPi.html
10. Orange Pi PC User Manual. [Електронний ресурс]. – Режим доступу: http://geekmatic.in.ua/pdf/OrangePi_PC_user_manual_v0.9.1.pdf
11. NXP PN532/C1 Near Field Communication (NFC) controller. [Електронний ресурс]. – Режим доступу: https://www.nxp.com/docs/en/nxp/data-sheets/PN532_C1.pdf
12. Google GSON Library. [Електронний ресурс]. – Режим доступу: https://github.com/google/gson
13. Vaadin Framework. [Електронний ресурс]. – Режим доступу: https://vaadin.com/
14. Spring Boot. [Електронний ресурс]. – Режим доступу: https://spring.io/

15. H2 Database Engine. [Електронний ресурс]. – Режим доступу: https://www.h2database.com/html/main.html

16. MySQL. [Електронний ресурс]. – Режим доступу: https://www.mysql.com/

17. Armbian Linux. [Електронний ресурс]. – Режим доступу: https://www.armbian.com/

18. Herbert Schildt, “Java The Complete Reference, Eleventh Edition”. McGraw-Hill, 2018. ISBN: 9781260440249

19. Kale, V. (2014). "Integration Technologies". Guide to Cloud Computing for Business and Technology Managers: From Distributed Computing to Cloudware Applications. CRC Press. pp. 107–134. ISBN 9781482219227

20. Samtani, G.; Sadhwani, D. (2013). "Integration Brokers and Web Services". In Clark, M.; Fletcher, P.; Hanson, J.J.; et al. (eds.). Web Services Business Strategies and Architectures. Apress. pp. 71–84. ISBN 9781430253563

21. Ejsmont, A. (2015). "Asynchronous Processing". Web Scalability for Startup Engineers. McGraw Hill Professional. pp. 275–276. ISBN 9780071843669

22. Java Persistence API [Електроннний ресурс] https://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

23. Linda DeMichiel, JSR 338: JavaTM Persistence API, Version 2.1 / Лінда Де-Міхаель, Oracle – 570 с. 23.

24. Hibernate ORM. Your relational data. Objectively. [Електроннний ресурс] http://hibernate.org/orm/

25. Hibernate - ORM Overview [Електроннний ресурс] https://www.tutorialspoint.com/hibernate/orm_overview.htm

ДОДАТОК 1

Програмно-апаратний комплекс системи контролю та управління доступом.

Специфікація

КР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6157_20Б

Аркушів 1

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ” КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6157_20Б	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ” КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6157_20Б 12-1	DBConnectionEvent.java DBUpdateService.java DoorService.java EventService.java GpioInputListener.java GPIO.java ModelController.java	Основні компоненти
УКР.НТУУ” КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6157_20Б 12-2	Додаток 2.doc	Опис програмного модуля

ДОДАТОК 2

Програмно-апаратний комплекс системи контролю та управління доступом.

Лістинг програмного модулю

КР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6157_20Б

Аркушів 7

Київ 2020

```
package com.denvys5.controller.service;

import com.denvys5.Settings;
import com.denvys5.model.Entity.Event;

import java.util.Date;

public class DBConnectionEvent extends Thread {
    private EventService eventService = EventService.getInstance();
    public void run(){
        eventService.addEvent(new Event(Settings.doorName, "System has started",
new Date()));
        while (!isInterrupted()){
            try {
                Thread.sleep(3600000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            eventService.addEvent(new Event(Settings.doorName, "System is running
as usual", new Date()));
        }
    }
}
package com.denvys5.controller.service;

import com.denvys5.Settings;
import com.denvys5.controller.service.socket.SocketEventListeners;
import com.denvys5.model.DAO;
import com.denvys5.model.Entity.Door;
import com.denvys5.model.Entity.Event;
import com.denvys5.model.dto.DoorDTO;
import io.netty.channel.ChannelHandlerContext;

import java.util.Date;

public class DBUpdateService {
    private static DBUpdateService ourInstance = new DBUpdateService();

    public static DBUpdateService getInstance() {
        return ourInstance;
    }

    private DBUpdateService() {
        eventService = EventService.getInstance();
        socketEventListeners = SocketEventListeners.getInstance();
        socketEventListeners.addReceiveMessageEventListener((args) ->
onDBRecieved((ChannelHandlerContext) args[0], args[1]));
    }
    private SocketEventListeners socketEventListeners;
    private EventService eventService;


    public void onDBRecieved(ChannelHandlerContext ctx, Object message){
        if(message instanceof DoorDTO){
            DoorDTO doorDTO = (DoorDTO) message;
            if(doorDTO.name.equals(Settings.doorName)){
                DAO.getInstance().wipeAndUpdateDB(new Door(doorDTO));
                eventService.addEvent(new Event(Settings.doorName, "DB has been
updated", new Date()));
            }else{
```

```
                eventService.addEvent(new Event(Settings.doorName, "Invalid DB
update", new Date()));
            }
        }
    }
}
package com.denvys5.controller.service;

import com.denvys5.Settings;
import com.denvys5.model.DAO;
import com.denvys5.model.Entity.Event;
import com.denvys5.model.Entity.NFC;
import com.denvys5.model.Entity.User;
import com.denvys5.model.dao.DoorDAO;
import com.denvys5.model.dao.NFCDAO;

import javax.persistence.NoResultException;
import javax.persistence.Query;
import java.util.Date;
import java.util.List;

public class DoorService {
    private static DoorService ourInstance = new DoorService();
    private DAO dao;
    private DoorDAO doorDAO;
    private NFCDAO nfcdao;
    private EventService eventService;
    private DBUpdateService dbUpdateService;

    public static DoorService getInstance() {
        return ourInstance;
    }

    private DoorService() {
        dao = DAO.getInstance();
        doorDAO = DoorDAO.getInstance();
        nfcdao = NFCDAO.getInstance();
        eventService = EventService.getInstance();
        eventService.start();
        dbUpdateService = DBUpdateService.getInstance();
    }


    public User getUserFromUID(String uid) throws NoResultException {
        NFC nfc = nfcdao.getNFCFromUID(uid);
        return nfc.getUser();
    }

    public void onDoorOpen(String uid){
        User user = getUserFromUID(uid);
        StringBuilder message = new StringBuilder();
        message.append("Door ")
                .append(Settings.doorName)
                .append(" is opened.");
        eventService.addEvent(new Event(user.getLogin(), message.toString(), new
Date()));
    }

    public boolean isNfcInDB(String uid) {
        Query query = dao.createQuery("from NFC where uid = :param");
```

```
        query.setParameter("param", uid);
        List<NFC> nfcs = query.getResultList();
        if(nfcs.isEmpty()) {
            eventService.addEvent(new Event(Settings.doorName, "Somebody, with
uid " + uid + " tried to access the Door", new Date()));
            return false;
        }
        return
nfcs.get(0).getUser().getDoors().contains(doorDAO.getDoorFromName(Settings.doorNa
me));
    }
}
package com.denvys5.controller.service;

import com.denvys5.controller.service.socket.ConnectionHolder;
import com.denvys5.model.Entity.Event;
import com.denvys5.model.dao.EventDAO;
import com.denvys5.model.dto.EventDTO;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;

public class EventService extends Thread {
    private static EventService ourInstance = new EventService();

    public static EventService getInstance() {
        return ourInstance;
    }

    private EventService() {
        eventDAO = EventDAO.getInstance();
        connectionHolder = ConnectionHolder.getInstance();
    }

    private EventDAO eventDAO;
    private ConnectionHolder connectionHolder;
    private List<EventDTO> eventsToSend = new CopyOnWriteArrayList<>();

    public void run(){
        while(!isInterrupted()){
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            if(eventsToSend.isEmpty())
                continue;

            connectionHolder.writeToChannel(new ArrayList<>(eventsToSend));
            eventsToSend.clear();
        }
    }

    public void addEvent(Event event){
        eventDAO.addEvent(event);
        eventsToSend.add(new EventDTO(event));
    }
}
package com.denvys5.controller.service;
```

```
import com.denvys5.controller.GPIO;
import com.pi4j.io.gpio.GpioPinDigitalInput;
import com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;

import java.io.IOException;

public class GpioInputListener extends Thread {

    private GpioPinDigitalInput arduinoInput;
    private GpioPinDigitalOutput redDiod;

    public GpioInputListener(GPIO gpio) {
        this.arduinoInput = gpio.getArduinoInput();
        this.redDiod = gpio.getRedDiod();
    }

    @Override
    public void run() {
        while(!isInterrupted()){
            if(arduinoInput.getState().isLow()){
                redDiod.setState(PinState.HIGH);
                System.out.println("Shutdown started");
                try {
                    Runtime.getRuntime().exec("sudo /sbin/shutdown -h now");
                } catch (IOException e) {
                    e.printStackTrace();
                }
                this.interrupt();
            }
        }
    }
}
package com.denvys5.controller;
import com.denvys5.controller.service.GpioInputListener;
import com.pi4j.io.gpio.*;
import com.pi4j.io.gpio.event.GpioPinDigitalStateChangeEvent;
import com.pi4j.io.gpio.event.GpioPinListenerDigital;
import com.pi4j.util.CommandArgumentParser;

import java.io.IOException;

public class GPIO extends Thread{
    private static GPIO ourInstance = new GPIO();

    public static GPIO getInstance() {
        return ourInstance;
    }

    private GPIO() {
    }
    private GpioPinDigitalOutput outputLock;
    private GpioPinDigitalOutput arduinoOut;
    private GpioPinDigitalOutput redDiod;
    private GpioPinDigitalOutput yellowDiod;
    private GpioPinDigitalOutput greenDiod;
    private GpioPinDigitalInput arduinoInput;
    private GpioInputListener arduinoInputListener;
    private GpioController gpio;
```

```
    public void run(){
        setUp();
    }

    public void setUp(){
        gpio = GpioFactory.getInstance();
//gerkon gpio31
        System.out.println("We are initializing gpio");

        Pin redDiodPin = CommandArgumentParser.getPin(OrangePiPin.class,
OrangePiPin.GPIO_22);
        redDiod = gpio.provisionDigitalOutputPin(redDiodPin, PinState.LOW);
        redDiod.setShutdownOptions(true, PinState.LOW);

        Pin yellowDiodPin = CommandArgumentParser.getPin(OrangePiPin.class,
OrangePiPin.GPIO_23);
        yellowDiod = gpio.provisionDigitalOutputPin(yellowDiodPin, PinState.LOW);
        yellowDiod.setShutdownOptions(true, PinState.LOW);

        Pin greenDiodPin = CommandArgumentParser.getPin(OrangePiPin.class,
OrangePiPin.GPIO_24);
        greenDiod = gpio.provisionDigitalOutputPin(greenDiodPin, PinState.LOW);
        greenDiod.setShutdownOptions(true, PinState.LOW);

        Pin pin = CommandArgumentParser.getPin(OrangePiPin.class,
OrangePiPin.GPIO_26);
//        Pin pin = CommandArgumentParser.getPin(OrangePiPin.class,
OrangePiPin.GPIO_01);
        outputLock = gpio.provisionDigitalOutputPin(pin, "My Output",
PinState.LOW);
        outputLock.setShutdownOptions(false, PinState.LOW);

        Pin ipin = CommandArgumentParser.getPin(OrangePiPin.class,
OrangePiPin.GPIO_12);
        PinPullResistance pull =
CommandArgumentParser.getPinPullResistance(PinPullResistance.PULL_DOWN);
        arduinoInput = gpio.provisionDigitalInputPin(ipin, pull);
        arduinoInputListener = new GpioInputListener(this);
        arduinoInputListener.start();
        arduinoInput.setShutdownOptions(true);

        System.out.println("We have initialized gpio");
        Pin tinypin = CommandArgumentParser.getPin(OrangePiPin.class,
OrangePiPin.GPIO_11);
        arduinoOut = gpio.provisionDigitalOutputPin(tinypin, PinState.LOW);
        arduinoOut.setShutdownOptions(true, PinState.LOW);
        arduinoOut.setState(PinState.HIGH);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        arduinoOut.setState(PinState.LOW);

        greenDiod.setState(PinState.HIGH);
        yellowDiod.setState(PinState.HIGH);
        redDiod.setState(PinState.LOW);


    }
```

```
    public void openDoor(){
        outputLock.pulse(400, true);// set second argument to 'true' use a
blocking call
    }

    public GpioPinDigitalOutput getRedDiod() {
        return redDiod;
    }

    public GpioPinDigitalOutput getYellowDiod() {
        return yellowDiod;
    }

    public GpioPinDigitalOutput getGreenDiod() {
        return greenDiod;
    }

    public GpioPinDigitalInput getArduinoInput() {
        return arduinoInput;
    }

    public void shutdown(){
        gpio.shutdown();
        this.interrupt();
    }

}
package com.denvys5.controller;


import com.denvys5.controller.service.DBConnectionEvent;
import com.denvys5.model.DAO;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class ModelController extends Thread{
    private static ModelController ourInstance = new ModelController();

    public static ModelController getInstance() {
        return ourInstance;
    }

    private ModelController() {
    }
    private EntityManagerFactory sessionFactory;
    private EntityManager entityManager;
    private boolean ready = false;

    private void setUp() {
        boolean connectionEstablished = false;
        while(!connectionEstablished){
            try{
                sessionFactory = Persistence.createEntityManagerFactory( "DB" );
                connectionEstablished = true;
            }catch (Exception e){
                try {
                    Thread.sleep(5000);
```

```
                } catch (InterruptedException e1) {
                    e1.printStackTrace();
                }
                e.printStackTrace();
                System.err.println("No connection to DB");
            }
        }
        System.out.println("Connection established");
        entityManager = sessionFactory.createEntityManager();
        DAO.setUpDependencyInjection(entityManager);
    }

    public boolean isReady(){
        return ready;
    }

    public void shutdown(){
        entityManager.close();
        sessionFactory.close();
    }

    public void run(){
        setUp();
        DBConnectionEvent dbConnectionEvent = new DBConnectionEvent();
        dbConnectionEvent.start();
        System.out.println("Model is fully initialized!");
        ready = true;
    }
}
```

ДОДАТОК 3

Програмно-апаратний комплекс системи контролю та управління доступом.

Опис програмного модулю

КР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТІ6157_20Б

Аркушів 8

Київ 2019

Анотація

Розділ містить опис програмної частини, яка слугує для забезпечення прийому виконання регламенту доступу користувачів системи після їх авторизації. Саме цей модуль приймає рішення надати доступ користувачеві до території, або приміщення.

ЗМІСТ

Загальні відомості

У додатку міститься частина коду, яка забезпечує виконання функцій авторизації користувача системи та надання доступу.

Програма була написана мовою Java у середовищі розробки JetBrains Intellij IDEA.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

У даній частині програми реалізуються такі функції:

1. Прийом даних авторизації користувача
2. Обробка даних авторизації
3. Прийняття рішення щодо надання доступу
4. Виклик відповідних апаратних інтерфейсів для надання доступу

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Модуль реалізований у формі набору класів. Він забезпечує зв’язок між користувачем і модулем авторизації та модулем забезпечення доступу, отримуючи дані, реагуючи на них та відправляючи відповідні команди у реальному часі. Тобто реалізує виконання регламенту доступу.

ТЕХНІЧНІ ЗАСОБИ ЩО ВИКОРИСТОВУЮТЬСЯ

Модуль розроблено у середовищі розробки JetBrains Intellij IDEA. При роботі програми використовуються такий пристрый як апаратна платформа модулю забезпечення доступу, що складається з таких пристроїв як мікрокомп’ютер OrangePi PC, мікроконтролер Atmega 328p, апаратна платформа Arduino Nano, та периферійні модулі.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Система забезпечує постійний контроль доступу в реальному часі, тому цей модуль запускається автоматично після запуску програмного забезпечення відповідного агенту і забезпечує виконання регламенту доступу.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними є інформація, яку надсилає користувач при авторизації. Це може запит, що містить у собі дані авторизації користувача.

Вихідними ж є вплив на модуль забезпечення доступу (наприклад, відкривання дверей) у разі якщо цього дозволяє регламент доступу.